AutoMax® Control Block Language

Instruction Manual J-3676-5

Rockwell Automation
Reliance Electric

The information in this user's manual is subject to change without notice.

WARNING

PROGRAMS INSERTED INTO THE PRODUCT SHOULD BE REVIEWED BY QUALIFIED PERSONNEL WHO ARE FAMILIAR WITH THE CONSTRUCTION AND OPERATION OF THE SYSTEM AND THE POTENTIAL HAZARDS INVOLVED. FAILURE TO OBSERVE THIS PRECAUTION COULD RESULT IN BODILY INJURY OR DAMAGE TO EQUIPMENT.

WARNING

ALL USERS MUST PROVIDE A HARDWIRED EMERGENCY STOP CIRCUIT OUTSIDE THE PROGRAMMABLE CONTROLLER CIRCUITRY. FAILURE TO OBSERVE THIS PRECAUTION COULD RESULT IN BODILY INJURY OR DAMAGE TO EQUIPMENT.

WARNING

ONLY QUALIFIED PERSONNEL MAY INSTALL, ADJUST, OPERATE, AND MAINTAIN THIS EQUIPMENT. READ AND UNDERSTAND THIS INSTRUCTION MANUAL BEFORE ANY WORK IS PERFORMED. FAILURE TO OBSERVE THIS PRECAUTION COULD RESULT IN BODILY INJURY OR DAMAGE TO EQUIPMENT.

WARNING

ONLY QUALIFIED PERSONNEL WHO HAVE READ AND UNDERSTOOD ALL APPLICABLE AUTOMAX INSTRUCTION MANUALS AND ARE THOROUGHLY FAMILIAR WITH THE PARTICULAR APPLICATION MAY UTILIZE THE ON-LINE PROGRAMMING OPTION PROVIDED IN THE AUTOMAX PROGRAMMING SOFTWARE. FAILURE TO OBSERVE THIS PRECAUTION COULD RESULT IN BODILY INJURY OR DAMAGE TO EQUIPMENT.

Table of Contents

Appendices

List of Figures

List of Tables

1.0 INTRODUCTION

The products described in this manual are manufactured or distributed by Reliance Electric Industrial Company.

The AutoMax Programming Executive software includes the software used to create and compile Control Block programs. This instruction manual describes AutoMax Control Block language for Version 2.0 and later AutoMax Programming Executive software.

Features that are either new or different from those in previous versions of the AutoMax Programming Executive software are so noted. Appendix A describes how to convert a Control Block task created with earlier versions of the Programming Executive to the current version.

1.1 Compatibility with Earlier Versions

Version 2.0 of the AutoMax Programming Executive requires AutoMax Processor M/N 57C430A or 57C431; Version 3.0 and later require AutoMax Processor M/N 57C430A, 57C431, or 57C435. M/N 57C430 cannot co-exist in the same rack with M/N 57C430A, 57C431, or 57C435. Refer to Appendix E for a listing of the AutoMax Processor modules that are compatible with Version 2 and later of the AutoMax Programming Executive software.

This instruction manual is organized as follows:

1.0	Introduction Where to find additional information Related hardware and software
2.0	General information about programming for AutoMax systems and Distributed Power systems
3.0	General information about programming in Control Block language
4.0	SCAN LOOP block
5.0-40.0	Control Blocks not dependent upon the SCAN LOOP block
41.0-53.0	Control Blocks dependent upon the SCAN LOOP block
54.0	Execution time estimates
Appendix A	Converting tasks created with previous versions of the Executive software to the current version
Appendix B	BASIC language statements and functions in Control Block tasks
Appendix C	Control Block functions supported in UDC Control Block tasks
Appendix D	BASIC language statements and functions supported in UDC Control Block tasks
Appendix E	AutoMax Processor compatibility with versions of the AutoMax Programming Executive

The thick black bar shown at the right-hand margin of this page will be used throughout this instruction manual to signify new or revised text or figures.

1.2 Additional Information

You should be familiar with the instruction manuals which describe your system configuration. This may include, but is not limited to, the following:

- J-3618 NORTON EDITOR REFERENCE MANUAL
- J-3649 AutoMax CONFIGURATION TASK INSTRUCTION MANUAL
- J-3650 AutoMax PROCESSOR INSTRUCTION MANUAL
- J-3675 AutoMax ENHANCED BASIC LANGUAGE INSTRUCTION MANUAL
- J-3677 AutoMax LADDER LOGIC INSTRUCTION MANUAL
- J2-3018 AutoMax REMOTE I/O SHARK INTERFACE INSTRUCTION MANUAL
- J2-3093 AutoMax Ladder Language Editor
- J2-3094 AutoMax Enhanced Ladder Language
- Your ReSource AutoMax PROGRAMMING EXECUTIVE INSTRUCTION MANUAL
- Your personal computer, DOS and Windows instruction manuals
- IEEE 518 GUIDE FOR THE INSTALLATION OF ELECTRICAL EQUIPMENT TO MINIMIZE ELECTRICAL NOISE INPUTS TO CONTROLLERS

1.3 Related Hardware and Software

The AutoMax Programming Executive software is used with the following hardware and software, which is sold separately.

1. M/N 57C430A, 57C431, or 57C435 AutoMax Processor.
2. IBM-compatible 80386-based personal computer running DOS version 3.1 or later. Version 4.0 and later Executive Software requires an 80486-based computer (or higher) running Windows 95.
3. M/N 61C127 RS-232C ReSource Interface Cable. This cable is used to connect the personal computer to the Processor module.
4. M/N 57C404A (and later) Network Communications module. This module is used to connect racks together as a network and supports communication with all racks on the network that contain 57C404A modules through a single Processor module. M/N 57C404 can be used to connect racks on a network; however, you cannot communicate over the network to the racks that contain M/N 57C404 Network modules. You must instead connect directly to the Processors in those racks.
5. M/N 57C413 or 57C423 Common Memory module. This module is used when there is more than one Processor module in the rack.
6. M/N 57C492 Battery Back-Up. This unit is used when there is a M/N 57C413 Common Memory module in the rack.

7. M/N 57C384 Battery Back-Up Cable. This cable is used with the Battery Back-Up unit.
8. M/N 57C554 AutoMax Remote I/O Shark Interface Module. This module is used to connect a Shark remote rack to the AutoMax Remote I/O network.
9. B/M 57552 Universal Drive Controller module. This module is used for drive control applications.
10. M/N 57C560 AutoMax PC3000 Processor/Scanner module. This module is a full-size ISA module that mounts in the personal computer.
11. M/N 57C565 AutoMax PC3000 Serial module. This module is a full-size ISA module that mounts in the personal computer.
12. M/N 57C570 Industrial AutoMax PC3000. This unit consists of a panel-mount, industrial grade enclosure containing an AutoMax PC3000 Processor/Scanner module, an AutoMax PC3000 Serial Interface module, and a power supply.

2.0 PROGRAMMING FOR AutoMax SYSTEMS

In AutoMax systems, application programs, also referred to as tasks, can be written in Ladder Logic/PC language, Control Block language, and Enhanced BASIC language. Control Block language is typically used for programming process control loops. It consists of BASIC statements and Control Block function calls. Refer to J-3675, J-3677, and J2-3094 for more information about Enhanced BASIC and Ladder Logic/PC programming.

In addition to multi-processing, AutoMax systems incorporate multi-tasking. This means that each AutoMax Processor (up to four) in a rack allows real-time concurrent operation of multiple application tasks.

Multi-tasking features allow the programmer's overall control scheme to be separated into individual tasks, each written in the programming language best suited to the task. This simplifies writing, check-out, and maintenance of programs; reduces overall execution time; and provides faster execution for critical tasks.

Programming in AutoMax systems consists of configuration, or defining the hardware, system-wide variables, and application tasks in that system, as well as application programming.

2.1 Configuration

Version 3.0 and Later Systems

If you are using AutoMax Version 3.0 or later, you define system-wide variables within the AutoMax Programming Executive. This eliminates the requirement to write a configuration task for the rack. See the AutoMax Programming Executive (J-3750) for information about configuring variables.

The information that follows is applicable if you are using AutoMax Version 2.1 or earlier. If you are using AutoMax 3.0 or later, you can skip over the remainder of this section and continue with 2.2.

Version 2.1 and Earlier Systems

AutoMax Version 2.1 and earlier requires a configuration task in order to define the following:

1. All tasks that will reside on the Processors in a rack.
2. All variables that equate to physical I/O in the system.
3. All other variables that must be accessible to all Processors in the rack.

One configuration task is required for each rack that contains at least one Processor. The configuration task must be loaded onto the Processor(s) in the rack before any application task can be executed because it contains information about the physical organization of the entire system.

The configuration task does not actually execute or run; it serves as a central storage location for system-wide information. Note that local variables, those variables that do not need to be accessible to more than one task, do not need to be defined in the configuration task. Refer to J-3649 for more information about configuration tasks.

2.2 AutoMax Application Tasks

AutoMax Processors allow real-time concurrent operation of multiple programs, or application tasks, on the same Processor. The tasks are executed on a priority basis and share all system data.

Each task operates on its own variables. The same variable names may be used in different tasks, but each variable is only recognized within the confines of its task unless it is specifically designated a COMMON variable. Changing local variable ABC% (designated LOCAL) in one task has no effect on variable ABC% in any other task.

Multi-tasking in a control application can be compared to driving a car. The programmer can think of the different functions required as separate tasks, each with its own priority.

In driving a car, the operator must monitor the speedometer, constantly adjust the pressure of his foot on the gas pedal, check the rearview mirror for other traffic, stay within the boundaries of his lane, etc., all while maintaining a true course to his destination. All of these functions have an importance or priority attached to them, with keeping the car on the road being the highest priority. Some tasks, like monitoring the gasoline gauge, require attention at infrequent intervals. Other tasks require constant monitoring and immediate action, such as avoiding obstacles on the road.

In a control application the Processor needs to be able to perform calculations necessary for executing a control scan loop, monitor an operator's console, log error messages to the console screen, etc. Of these tasks, executing the main control loop is obviously the most important, while logging error messages is the least important. Multi-tasking allows the control application to be broken down into such tasks, with their execution being dependent upon specified "events," such as an interrupt, operator input, or the expiration of a time interval.

The following table is a representation of typical tasks found in a control application and the kind of event that might trigger each.

Task	Triggering Event
Execute main control loop	Expiration of a hardware timer that indicates the interval at which to begin a new scan
Respond to external I/O input	Generation of a hardware interrupt by an input module
Read operator data	Input to an operator panel
Log information	Expiration of a software timer

Each of these tasks would be assigned a priority level (either in the specific configuration task for the rack, or in later versions of the Programming Executive software, through the configuration option). The priority determines which task should run at any particular instant. The more important the task, the higher the task priority.

2.3 Univeral Drive Control Application Tasks

Universal Drive Control (UDC) Control Block tasks are used exclusively for drive control applications. Each UDC module in an AutoMax rack can run up to 2 independent Control Block tasks (one for Drive A and another for Drive B) that provide speed loop control. UDC Control Block tasks share all system data. These tasks are not assigned a priority.

Each task operates on its own variables. The same variable names may be used in different tasks, but each variable is recognized only within the confines of its task unless it is specifically designated a COMMON variable. Changing local variable ABC% (designated LOCAL) in one task has no effect on variable ABC% in any other task.

UDC Control Block tasks can use most, but not all, of the Control Block function calls in the AutoMax Control Block language. See Appendix C for a list of the Control Block functions that are allowed in UDC Control Block tasks. Also, note that the descriptions of Control Block functions in this manual indicate whether a particular function can be used in UDC Control Block tasks.

UDC Control Block tasks can be up to 20 Kbytes in size. These tasks are run at a fixed tick rate of .5 milliseconds. The maximum scan time allowed (set by using the SCAN LOOP function) is 20 ticks (10 milliseconds). Note that if a UDC module will be running two tasks, both must be assigned the same scan time. See section 53.0 for information regarding how to estimate execution time for a UDC Control Block task.

2.4 AutoMax Programming Conventions

This section describes programming conventions that apply to all Configuration, BASIC, Control Block, and Ladder Logic/PC tasks.

2.4.1 Naming

All task names are limited to 8 characters. The initial character must always be a letter. Only letters (A-Z), underscores (_), and numbers (0-9) are permitted. Spaces and other characters are not permitted in task names. The file extension is used to identify the task. Extension .CNF identifies configuration tasks (used only in Programming Executive Version 2.1 and earlier). .BAS is used for BASIC tasks. AutoMax Control Block tasks use extension .BLK. UDC Control Block tasks also use extention .BLK. PC/Ladder Logic tasks have a .PC extension.

Variable names in BASIC, AutoMax Control Block, and UDC Control Block tasks are limited to 16 characters. Variable names in Ladder Logic/PC tasks are limited to 14 characters (16 characters in V4.0 and later). The initial character of variable names must always be a letter or an underscore. Only letters (A-Z), numbers (0-9) and the

underscore (_) are permitted within the variable name. Spaces and other characters are not permitted in variable names. Terminating characters used to specify the variable type (see section 2.4.3) are not included in the size limits.

2.4.2 Constants

A constant, also known as a literal, is a fixed value that is not associated with a variable name. Listed below are the five types of constants that can be used in AutoMax, along with their size limitations:

1. **Integer Constants (Integers and Double Integers)**

   Single integer value range is +32767 to −32768 with no fractional part. Double integer value range is +2147483647 to −2147483648 with no fractional part.

2. **Hexadecimal Constants**

   Hexadecimal constants are integers in base 16, or "hex" format. The allowable range of hexadecimal constants is 0 to 0FFFFFFFFH. Hexadecimal constants are not sign-extended when they are stored. Leading zeroes are used to fill in any of the hex digits not specified. This means that numbers must be entered as 2's complement signed numbers. For example, if you enter 0F371H, it will be stored as 0000F371H, not as 0FFFFF371H.

   A hexadecimal number has three parts:

   {0}NNNNNNNNH

   where: 0 = required only when the first digit of the hexadecimal number is an alpha character (A-F) NNNNNNNN = the hexadecimal number H = specifies that the number is in hexadecimal format; always required

3. **Real Constants**

   Real constants are decimal values. The value can be in the following ranges:

   9.2233717 x 10**18 > positive value > 5.4210107 x 10 ** (−20)
   −9.2233717 x 10**18 > negative value > −2.7105054 x 10 ** (−20)

   Only eight digits of significance are accepted. The format for entering real constants is as follows:
   {sign}{digits}{.}{E}{sign}{digits}
   For example: −1234.5678E+11

   Use scientific notation to enter large numbers. Use double asterisks to indicate exponentiation.

4. **String Constants**

   String constants are sequences of alphanumeric and other printable characters. Line terminators (<CR>) are not allowed. String constants must be enclosed either in single or double quotes. If one type of quotes is used in the sequence itself, the other type must be used to enclose the sequence. String constants may be up to 132 characters long.

5. **Boolean Constants**

   There are four boolean constants: TRUE, ON, FALSE, and OFF.

2.4.3 Variables

A variable is a named location that represents a value or a physical I/O location. A variable may be either a simple variable or a subscripted (array) element. Depending upon the operations specified in the application task, the value of a variable may change from line to line. BASIC tasks use the most recently assigned value of a variable when performing calculations. Control Block and Ladder Logic/PC tasks latch the value of common double integer, integer, and boolean variables at the beginning of the task scan to ensure that external inputs will not change state during a scan.

The data type of a variable determines the type of information stored in that variable. For variables that contain numeric information, the data type also determines the range of values that may be stored in the variable. Values that are not in the allowable range will cause an error when the task is compiled or when it is put into run, depending upon the type of error. The data type of a variable is specified with a terminating character for four of the five types of variables. The fifth type, real variables, do not have a terminating character.

1. **Long Integer or Double Integer Variables**

   Used to store 32 bits. The value can be in the range +2147483647 to −2147483648 with no fractional part. The terminating character is !. If you assign a real number (see #3 below) to an integer variable, the fractional part will be truncated.

2. **Integer or Single Integer Variables**

   Used to store 16 bits. The value can be in the range +32767 to −32768 with no fractional part. The terminating character is %. If you assign a real number (see #3 below) to the variable, the fractional part will be truncated. Note that all internal integer calculations are in double precision, or 32 bits.

3. **Real Variables**

   Used to store a decimal value. The value can be in the following ranges:

   9.2233717 x 10**18 > positive value > 5.4210107 x 10 **(−20)
   −9.2233717 x 10**18 > negative value > −2.7105054 x 10 ** (−20)

   There is no terminating character for real variables. Use scientific notation to enter large numbers. Use double asterisks to indicate exponentiation.

   Only eight digits of significance are accepted. The format for entering real constants is as follows:
   {sign}{digits}{.}{E}{sign}{digits}
   For example: −1234.5678E+11

4. **Boolean Variables**

   Used to store the status of 1 bit. The value can be either TRUE or FALSE or ON or OFF. The terminating character is @. Note that in BASIC tasks, the inverted sense (negative of the current

state) of a boolean variable is indicated with NOT. In Control Block language only, the inverted sense of a boolean variable can be indicated by entering a minus sign in front of the name when referencing it in the application task. For example, STATUS@ indicates the normal sense of variable STATUS@, while −STATUS@ would indicate the inverted sense of variable STATUS@. Note that inverted booleans cannot be assigned to outputs from control blocks.

5. **String Variables**

   Used to store any alphanumeric sequence of printable characters, including spaces, tabs, and special characters. The terminating character is $.

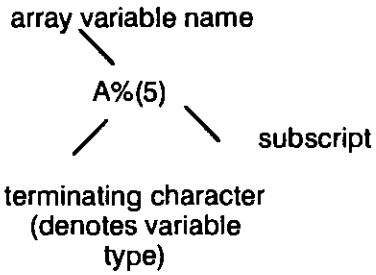
   The sequence in a string variable cannot include a line terminator (<CR>). When defined, the sequence must be enclosed either in single or double quotes. If one type of quotes is used in the sequence itself, the other type must be used to enclose the sequence.

   Version 1.0 Executive software allowed a fixed maximum length of 31 characters for string variables. Version 2.0 and later allow string variables of variable length, from 1 to 255 characters. To specify the maximum size of a string variable, add a colon and a number (1-255) immediately after the $ character. For example, defining A$:50 as a local variable in an application task will reserve space for 50 characters. Note that if no length is specified, the default length is 31.

2.4.4 Arrays

Array variables are used to store a collection of data all of the same data type. Arrays are permitted for all data types. Arrays are limited to four dimensions, or subscripts. The number of elements in each dimension is limited to 65535. The term array is used to denote the entire collection of data. Each item in the array is known as an element.

Array variables are specified by adding a subscript(s) after the variable name which includes the appropriate terminating character to denote the type of data stored in the array. The terminating character is followed by a left parenthesis (or bracket), the subscript(s), and a right parenthesis (or bracket). Multiple subscripts are separated by commas. Note that subscripts can be integer constants as well as arithmetic expressions that result in integer values.



An array with one dimension, i.e., one subscript, is said to be one-dimensional. An array with two subscripts is said to be two-dimensional, etc. The first element in each dimension of the array is always element 0. Therefore, the total number of elements in

each dimension of the array is always one more than the largest subscript.

Example 1 - One-dimensional array

A%

0	1	2	3	4	5
185	2	53	79	99	122

value of A

Example 2 - Two-dimensional array

B% (6, 3)

B%

	0	1	2	3	4	5	6
0	185	2	53	79	99	122	40
1	70	36	46	31	34	85	6
2	77	73	21	365	476	51	47
3	18	23	53	342	39	224	107

In the case of string arrays, version 1.0 Executive software always allocated the maximum amount of memory for each element in the array, regardless of whether the string stored in that element was of the maximum length, 31 characters. Version 2.0 (and later) Executive software allows the programmer to specify the maximum size of elements in the array, from 1 to 255 characters.

To specify the maximum size of string variables in an array, add a colon and a number (1-255) immediately after the $ character when declaring the variable in an application task or defining it during configuration. For example, defining A$:10(20) as a local variable in an application task allocates space for 21 string values of 10 characters each. Note that if no length is specified in the initial array reference, the default maximum is 31.

To define an array that will be common, i.e., accessible to all tasks in the rack, you need to first define the variable. If you are using AutoMax Version 2.1 or earlier, this is done with a MEMDEF or NVMEMDEF statement in the configuration task for the rack. If you are using AutoMax Version 3.0 or later, common variables are defined within the Programming Executive. For example, ARRAY1@(10) will allocate space for 11 boolean variables. Then, in an application task for the rack, you declare the array a COMMON variable as follows:

```
COMMON ARRAY1@(10).
```

Each element of the array that will be used in the task can be defined with LET statements as follows:

```
LET ARRAY1@(0) = TRUE
```

(boolean values can only be TRUE/FALSE or ON/OFF). Other application tasks in the rack can access the value in variable ARRAY1@(0) simply by declaring it a COMMON variable.

2.4.5 Variable Control Types

The control type of a variable refers to the way the variable is declared or defined in the rack configuration and application tasks. There are two control variable types in AutoMax systems, local and common.

1. **Local**

   Local variables are variables that are not defined in the configuration for the rack and are therefore accessible only to the application task in which they are defined. BASIC and Control Block tasks must define the variables with a BASIC LOCAL statement. For Ladder Logic/PC tasks, the editor prompts for whether the variable is local or common when the task is being created.

   In BASIC and Control Block tasks, local variables can be defined as tunable. Tunables are variables whose value can be tuned, i.e., changed within limits, by the operator through the On-Line menu of the Executive software. The value of tunable variables can not be changed by application tasks. BASIC and Control Block tasks must define tunable variables with a variation of the BASIC LOCAL statement that includes the tuning parameters. Ladder Logic/PC tasks cannot use tunable variables.

   The value of local variables at the time of initial task installation is always 0. The effect of a STOP ALL or a power failure on variable values in the rack depends on the variable type. Local tunable variable values in both AutoMax and UDC application tasks is always retained. Local variable values are retained for AutoMax tasks, but not for UDC tasks.

   AutoMax Processors will retain the last values of all local variables. UDC modules will retain the variable values for the following: parameter configuration data, UDC test switch information, and D/A setup configuration. The variable values of the following input data will also be retained: feedback registers, UDC-PMI communication status registers, and UDC task error log information. UDC modules will NOT retain local variable values and data found in the following registers, which are considered outputs: command registers, application registers, the ISCR (interrupt status and control register), scans per interrupt register, and scans per interrupt counter register. See the AutoMax Programming Executive for more information on the STOP ALL and system re-initialization conditions.

2. **Common**

   Common variables are variables that are defined in the configuration for the rack and are therefore accessible to all application tasks in the rack. There are two types of common variables, those that refer to memory locations, and those that refer to actual physical I/O locations. The two types are defined differently in the configuration task for the rack.

   Common memory variables can be of any data type. They may be read to or written from. Common I/O variables are long integer, integer, or boolean variables that represent actual physical I/O locations. Common I/O variables that represent inputs may be read but not written to. I/O variables that represent outputs may be read or written to.

All BASIC and Control Block tasks that need to access common variables can do so by using the BASIC statement COMMON (or GLOBAL). For Ladder Logic/PC tasks, the editor prompts for whether the variable is local or common when the task is being created. At least one task in the rack should also initialize common memory variables, i.e., assign values to them, if they need to be at a known state other than 0.

The value of common variables at the time of initial task installation depends upon whether the variable references memory or physical I/O locations. Common memory variables are always 0 at task installation. Common I/O variables that represent outputs are always 0. Common I/O variables that represent inputs are always at their actual state.

After a STOP ALL condition or a power failure followed by a system-restart, common memory variables that are defined as volatile memory in the configuration for the rack are 0. Common memory variables that are defined as non-volatile memory in the configuration retain their last value. Common variables that represent I/O locations are at 0 for outputs and at their actual state for inputs. Note that the UDC dual port memory is treated like I/O variables. See the AutoMax Programming Executive for more information on the STOP ALL and system-restart conditions.

2.4.6 Pre-defined Common Memory Variables

The following common memory variables are pre-defined for every rack. However, they do not appear on the form for common memory variables. You must enter these variable names on the form if you want to use these variables in application tasks.

AUTORUNSTATUS@ - True when AUTO RUN is enabled for the rack; false if AUTO RUN is not enabled

FORCINGSTATUS@ - True when a variable is forced in the rack; false when no variables are forced in the rack

BATTERYSTATUS0@ - True when the on-board battery of the Processor module or Common Memory module in slot 0 is OK

BATTERYSTATUS1@ - " " " " " " " " " " " " " 1 " " "
BATTERYSTATUS2@ - " " " " " " " " " " " " " 2 " " "
BATTERYSTATUS3@ - " " " " " " " " " " " " " 3 " " "
BATTERYSTATUS4@ - " " " " " " " " " " " " " 4 " " "

3.0 CONTROL BLOCK PROGRAMMING

AutoMax Control Block is a high-level language used for programming process control loops. It consists of BASIC language statements and function calls, or blocks, that are commonly used in the design of control systems. These functions have a graphic form that is used during the design and check-out phase of a project, as well as a textual form that is used by the programmer for creating the program. The graphic form includes an abbreviated symbol for each input and output. An arrow pointing into a block indicates a required input parameter. An arrow pointing out of a block indicates a required output parameter.

A file that contains templates of the textual form of all blocks is loaded onto the hard disk of the personal computer when the Executive software is installed. The file is named "TEMPLATE.BLK" and is stored in the AutoMax installation directory. You can access the file when creating Control Block tasks using a text editor.

TEMPLATE.BLK can be used when creating either AutoMax Control Block tasks or UDC Control Block tasks. AutoMax PC3000 Control Block tasks are identical to AutoMax Control Block tasks. Note that some of the blocks listed in TEMPLATE.BLK cannot be used in UDC Control Block tasks. The text editor will not prevent you from using an illegal block, but the compiler will catch the error. The task will not compile successfully until any illegal blocks are removed. Refer to Appendix C for a list of the Control Block functions supported in UDC Control Block tasks.

3.1 Format of Control Blocks

A control block, or function call, consists of the following:

a line number: positive integer in the range 1-32767 inclusive

the keyword CALL

the function name that identifies the function

the parameter list, which contains the inputs and outputs for the function.

Note that there are three types of inputs: optional, required, and default. Optional means that the value or parameter does not need to be entered. Required means that the value/parameter must be entered. Default means that the value/parameter does not need to be entered by the user, but that the value is still required in order to execute the function and, if the user does not enter a value, the default value will be used in executing the function.

3.2 Task Execution

Each AutoMax Control Block task or UDC Control Block task must include a SCAN LOOP block. The SCAN LOOP block specifies the periodic interval at which the task will execute. The SCAN LOOP block must be the first block called in a Control Block task. Any BASIC statement that could cause the task to be suspended is not permitted. The BASIC statement END must be executed at the end of each scan of a Control Block task. See Table 4 and 5, Maximum

Execution Time Summary, for information on execution times for all blocks. Note that UDC Control Block tasks can use only the blocks listed in Table 5. See the AutoMax Programming Executive instruction manual for more information on task execution.

3.3 Variable Definition and Initialization

All variables used in the task must be defined using the BASIC statements COMMON or LOCAL at the beginning of the scan before the SCAN LOOP block. All variables that must be at a known state or at a non-zero state must also be initialized in the task, i.e., a value must be assigned to them, before the SCAN LOOP block. Variables are initialized using the BASIC statement LET or the BASIC relational operator "=".

At the beginning of each scan, the values of the following are latched, i.e., read into a local buffer for reference throughout the scan: all simple (non-subscripted) common integer, double integer, boolean variables. This ensures that external inputs will not change state during a scan. Reals, strings, and array variables of any data type are not latched.

As the task executes, each block will read/write simple common variables of integer, double integer, and boolean type to this local buffer. Each block that references these variables will always see the value of those variables as currently stored in the buffer, even if another task modifies the actual state of the variables in question. At the end of the Control Block task, all buffered variables are examined to see if they are different from their initial state. If so, the new value is written to the actual variable.

Note that BASIC statements, whether in a BASIC task, an AutoMax Control Block task, or a UDC Control Block task, always reference the current value/state of all common variables. Variables are not buffered for BASIC statements. Instead, as they are referenced via the BASIC statement, they are read from and written to their actual location, whether it is a common memory location or a common I/O point.

For example, Control Block task ABC references variable COMM_VAR% from several control block statements within the task. Task XYZ, which is in the same Processor module or another Processor module in the same rack, writes to COMM_VAR%. At the start of the scan of ABC, the value of COMM_VAR% is 100 and is read and stored in the local buffer of ABC. Task XYZ runs and changes the value of COMM_VAR% to 299 before task ABC has completed its scan. Since the buffered value is used, all control block statements that reference COMM_VAR% will see 100 rather than 299 as the value of COMM_VAR%.

WARNING

IF BASIC STATEMENTS ARE USED WITHIN A CONTROL BLOCK TASK, ENSURE THAT A CONFLICT DOES NOT ARISE BETWEEN A CONTROL BLOCK'S USE OF A PARTICULAR COMMON VARIABLE AND A BASIC STATEMENT'S USE OF THE SAME COMMON VARIABLE. A CONFLICT MAY ARISE IF A BASIC STATEMENT REFERENCES A COMMON VARIABLE ALSO REFERENCED BY A CONTROL BLOCK STATEMENT WITHIN THE TASK. FAILURE TO OBSERVE THESE PRECAUTIONS COULD RESULT IN BODILY INJURY OR IN DAMAGE TO OR DESTRUCTION OF THE EQUIPMENT.

In the example below, if a BASIC statement produces variable COMM_VAR%, subsequently referenced as an input to a control block statement, the DIFFERENCE block will not see the new state of COMM_VAR% produced by statement 101. Instead, it will see the state produced during the last scan of the task. This is because the DIFFERENCE block will obtain the value of COMM_VAR% from the LOCAL buffer and the BASIC statement at line 101 will write the new value for COMM_VAR% to its common location. This value will then be read into the task's LOCAL buffer at the start of the next scan.

```
101 COMM_VAR% = LOCAL_A% + LOCAL + C%
 .
 .
 .
204 CALL DIFFERENCE(INPUT1 = COMM_VAR%,      &
      INPUT2 = FDBK%, OUTPUT = CNTL_ERR%)
```

Control Block tasks in which BASIC statements and control blocks reference the same simple common integer, double integer, and boolean variables must be constructed taking into account the fact that BASIC statements will always reference the actual current value of the variable, whereas control blocks will always reference the value of the variable that is in the buffer.

Potential problems can be avoided if both BASIC statements and control blocks in the task use local variables, or if common variables used in BASIC statements are not used by any control blocks.

Note that is you define both an integer and bits within that integer, the variable values are stored in the Control Block task buffer independently. If you write to the integer variable in the buffer (i.e., with a Control Block statement), you do not affect the integer's bits within the buffer. If you use BASIC statements to write to the integer or bits within the integer, you are referencing the actual variable values, not the values in the buffer. If you want to change the value of an integer and the bits within it, use BASIC statements to do so.

See Appendix B for the BASIC statements that can be used in AutoMax Control Block tasks. See Appendix D for the BASIC statements that can be used in UDC Control Block tasks. See the Enhanced BASIC Language instruction manual, J-3675, for error codes that can occur when application tasks are compiled and when they are put into RUN.

4.0 SCAN LOOP

This function can be used in AutoMax Control Block tasks and UDC Control Block tasks.

Function

The SCAN LOOP block is used to define the time intervals at which the task will execute. The periodic interval can be controlled by either the AutoMax Processor's or UDC module's real time clock or a system EVENT (for AutoMax Processor only) that is defined prior to the call of this block.

Note that this block is required in every AutoMax Control Block task or UDC Control Block task. The SCAN LOOP block must be the first block called. All definition of variables and initialization required must occur before this block is called.

This block also performs the function of latching all simple common integer (%), double integer (!), and boolean (@) variables defined within the task each time the task runs.

Program Statement

CALL SCAN_LOOP(TICKS=ticks%, EVENT=event name)

Inputs

TICKS =

Specifies the periodic interval of execution for the control loop, type INTEGER. For versions of the AutoMax Programming Executive earlier than 3.1, a tick is defined as 5.5 milliseconds. The AutoMax Programming Executive Version 3.1 (and later) allows you to assign a tick rate ranging from 0.5 milliseconds to 10 milliseconds for each AutoMax Processor. The default tick rate is 5.5 milliseconds.The tick rate for UDC modules is fixed at .5 milliseconds. The maximum number of ticks for UDC Control Block tasks is 20. See J-3750 or J2-3045 for more information. This parameter must be specified and must be specified as a literal value only (variable name not accepted).

EVENT =

Specifies the name of the previously defined (through a BASIC statement) hardware event that causes the task to run. This parameter is optional. If specified, the task will execute based on the event. If not specified, the task will execute based on the system clock. **This parameter cannot be specified for an AutoMax PC3000 Control Block task or a UDC Control Block task.**

Even if the EVENT input is programmed, the TICKS input must still be programmed. The TICKS input is used to define the scan period of the task that is required when making scan-time dependent coefficient calculations for frequency-based control blocks

(PROP_INT, LAG, etc.).

```
100 COMMON ISCR%
          .
          .
          .
500 EVENT NAME=START_TASK,    &
    INTERRUPT_STATUS=ISCR%, TIMEOUT=6
          .
          .
          .
1000 CALL SCAN_LOOP(TICKS=4, EVENT=START_TASK)
```

The EVENT input parameter is intended to be used for AutoMax Processor control loop tasks having a feedback parameter obtained from an interrupting I/O module. There are presently four input modules that have this ability: the Resolver Input Module (57C411) the 32 Channel Input Module (57C419), the 2 Channel Analog Input Module (57C409), and the Pulsetach Module (57C421). These modules have the ability to latch the input data when the interrupt is generated. This effectively "freezes" the data in time until the task that uses it gets its turn to execute and read it from the input module.

The period at which the data is latched and the interrupt is generated is programmable. Refer to individual I/O module documentation for more information.

Note that the BASIC statement GOTO is not permitted to reference a SCAN_LOOP block or any program line that precedes this block. A BASIC END statement must be executed every scan.

5.0 ABSOLUTE VALUE

This function can be used in AutoMax Control Block tasks and UDC Control Block tasks.

Function

OUTPUT = | INPUT |

Program Statement

```
CALL ABSOLUTE_VALUE(                    &
    INPUT = input%,                     &
    OUTPUT = output%)
```

Inputs

I (INPUT) =

INTEGER signal input. This parameter must be specified as a variable name only (literal value not accepted).

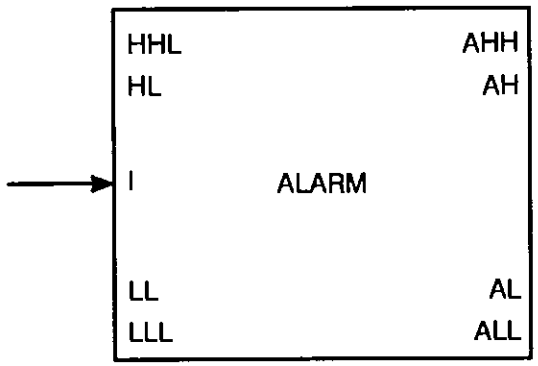
Outputs

O (OUTPUT) =

INTEGER signal output. This parameter must be specified.

6.0 ALARM

This function can be used in AutoMax Control Block tasks and UDC Control Block tasks.



Function

If INPUT is equal to or exceeds any of the alarm limits, the proper ALARM output(s) are set TRUE.

Program Statement

```
CALL ALARM(INPUT=input%,                          &
     HIGH_LIMIT=high_limit%,                      &
     LOW_LIMIT=low_limit%,                        &
     HIGH_HIGH_LIMIT=high_high_limit%,            &
     LOW_LOW_LIMIT=low_low_limit%,                &
     ALARM_HIGH=alarm_high@,                      &
     ALARM_LOW=alarm_low@,                        &
     ALARM_HIGH_HIGH=alarm_high_high@,            &
     ALARM_LOW_LOW=alarm_low_low@)
```

Inputs

I (INPUT) =

Signal input, type INTEGER. This parameter must be specified as a numeric symbol only (literal value not accepted).

HL (HIGH_LIMIT) =

Alarm high limit, type INTEGER. This is an optional parameter. It sets the high limit alarm value. The default is +32767.

LL (LOW_LIMIT) =

Alarm low limit, type INTEGER. This is an optional parameter. It sets the low limit alarm value. The default is -32768.

HHL (HIGH_HIGH_LIMIT) =

Alarm high-high limit, type INTEGER. This is an optional parameter. It sets the high_high limit alarm value. The default is +32767.

LLL (LOW_LOW_LIMIT) =

Alarm low-low limit, type INTEGER. This is an optional parameter. It sets the low-low limit alarm value. The default is -32768.

Outputs

AH (ALARM_HIGH) =

Alarm high output, type BOOLEAN. This parameter is optional. If INPUT is greater than or equal to HIGH_LIMIT, then ALARM_HIGH output is set TRUE.

AL (ALARM_LOW) =

Alarm low output, type BOOLEAN. This parameter is optional. If INPUT is less than or equal to LOW_LIMIT, then ALARM_LOW output is set TRUE.

AHH (ALARM_HIGH_HIGH) =

Alarm high-high output, type BOOLEAN. This parameter is optional. If INPUT is greater than or equal to HIGH_HIGH_LIMIT, then ALARM_HIGH_HIGH output is set TRUE.

ALL (ALARM_LOW_LOW) =

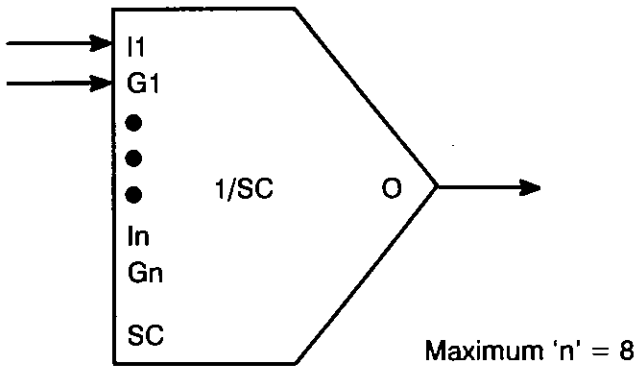
Alarm low-low output, type BOOLEAN. This parameter is optional. If INPUT is less than or equal to LOW_LOW_LIMIT, then ALARM_LOW_LOW output is set TRUE.

Notes

1. The order in which the outputs are programmed is unimportant. However, a minimum of one output must be programmed. If this requirement is not met, a compilation error will occur.

7.0 AMPLIFIER

This function can be used in AutoMax Control Block tasks and UDC Control Block tasks.



Function

OUTPUT = (INPUT1 * GAIN1+...INPUTn * GAINn)/SCALE)

Program Statement

```
CALL AMPLIFIER(INPUT1=input1%,                    &
     GAIN1 = gain1%,...,                          &
     INPUTn = inputn%,                            &
     GAINn = gainn%,                              &
     SCALE = scale%,                              &
     OUTPUT = output%)
```

Inputs

I1 (INPUT) =

INTEGER signal type 1. This parameter must be specified.

G1 (GAIN) =

INTEGER gain 1. This parameter must be specified.

In (INPUTn) =

INTEGER signal input n. This is an optional parameter. A maximum of 8 input/gain pairs can be specified.

Gn (GAINn) =

INTEGER gain n. This is an optional parameter; however, a GAIN must be specified for each INPUT.

SC (SCALE) =

INTEGER scale factor. The default for this parameter is 10000.

Outputs

O (OUTPUT) =

INTEGER signal output. This parameter must be specified.

Notes

1. The order in which (INPUTn) and (GAINn) pairs are entered is not important. However, it is required that, for "m" channels used, channels 1 through "m" be programmed. In other words, the channels used must be contiguous beginning with channel 1. A compilation error will occur if this requirement is not met.

The following is a correct statement:

```
CALL AMPLIFIER(                                          &
    INPUT1 = INA%,GAIN1 = GAINA%,                        &
    INPUT2 = INB%,GAIN2 = GAINB%,                        &
    INPUT3 = INC%,GAIN3 = GAINC%,                        &
    INPUT4 = IND%,GAIN4 = GAIND%,                        &
    SCALE = scale%,OUTPUT = output%)
```

The following is also correct, illustrating that the order of entry is not important:

```
CALL AMPLIFIER(                                          &
    INPUT2 = INB%,GAIN2 = GAINB%,                        &
    INPUT4 = IND%,GAIN4 = GAIND%,                        &
    INPUT3 = INC%,GAIN3 = GAINC%,                        &
    INPUT1 = INA%,GAIN1 = GAINA%,                        &
    SCALE = scale%,OUTPUT = output%)
```

The following is an incorrect statement because the channels are not contiguous:

```
CALL AMPLIFIER(                                          &
    INPUT1 = INA%,GAIN1 = GAINA%,                        &
    INPUT3 = INC%,GAIN3 = GAINC%,                        &
    INPUT4 = IND%,GAIN4 = GAIND%,                        &
    SCALE = scale%,OUTPUT = output%)
```

2. Overflow handling: During block execution, (INPUTn) and (GAINn) pairs are read and processed in sequential order beginning with (INPUT1) and (GAIN1) until a null pair is found. As the sum of the products from (INPUTn*GAINn) is computed and an overflow of 32 bits occurs (sum exceeds +2147483647 or −2147483648), the sum will be clamped to +2147483647 or −2147483648, an error will be logged, and the OUTPUT will be clamped to +32767 or −32768. Once a 32-bit overflow occurs, all remaining input channels will be ignored.

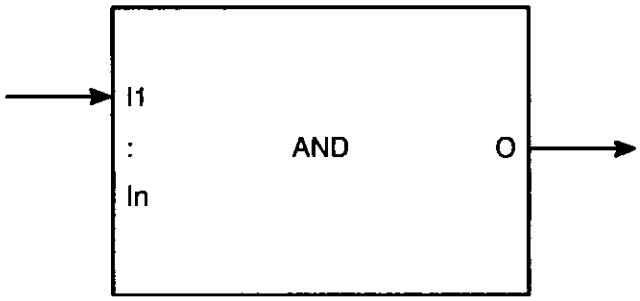
   When all consecutive sequential input channels are processed, the 32-bit sum is divided by SCALE, producing OUTPUT as a 16-bit result. If an overflow occurs on the divide, that is, the result exceeds 16 bits +32767 or −32768, an error will be logged and the result will be clamped to +32767 or −32768, producing OUTPUT.

3. Multiplying a positive input by −1 will result in that input being subtracted from the other inputs.

4. Setting SCALE <0 will effectively invert the sign of the output.

8.0 AND

This function can be used in AutoMax Control Block tasks and UDC Control Block tasks.



Maximum 'n' = 8

Function

OUTPUT = INPUT1 and ... INPUTn

Program Statement

```
CALL AND (INPUT1=input1@,                    &
    INPUTn=inputn@,                          &
    OUTPUT=output@)
```

Inputs

I1 (INPUT1) =

Signal input 1, type BOOLEAN. This parameter must be specified.

In (INPUTn) =

Signal input n, type BOOLEAN. This is an optional parameter. A maximum of 8 inputs can be specified.

Outputs

O (OUTPUT) =

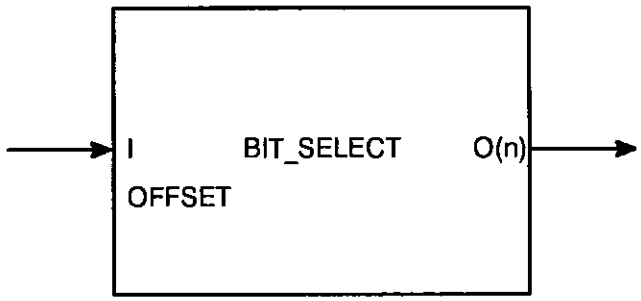
Data output, type BOOLEAN. This parameter must be specified. If all specified inputs are TRUE then OUTPUT will be TRUE.

Notes

1. The inputs programmed must be contiguous beginning with input 1. A compilation error will occur if this requirement is not met.

9.0 BIT SELECT

This function can be used in AutoMax Control Block tasks only. It cannot be used in UDC Control Block tasks.



'n' = 0 . . . 15

Function

If (INPUT - OFFSET) >= 0 and (INPUT - OFFSET) =< 15
then OUTPUT(INPUT - OFFSET) is set TRUE

All other outputs(n) are set FALSE

else

All outputs(n) are set FALSE

Program Statement

```
CALL BIT_SELECT(INPUT=input%, OFFSET=offset%,                    &
     OUTPUT0=output0@, ...OUTPUTn=outputn@)
```

Inputs

I (INPUT) =

Signal input, type INTEGER. This parameter must be specified.

OFFSET (OFFSET) =

Offset input value, type INTEGER. This parameter is optional. The default value is zero if not specified. This value is subtracted from INPUT to calculate the index (0-15) for OUTPUT(n).

Outputs

O(n) (OUTPUT) =

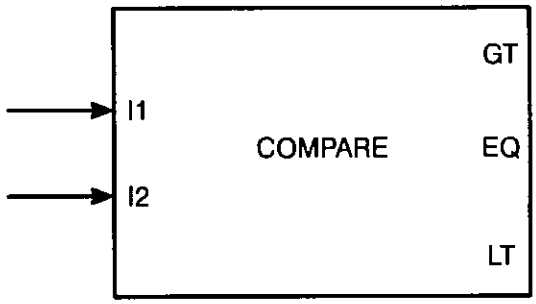
Data output 0...15, type BOOLEAN. The outputs can be specified in any order.

Notes

1. The order in which the outputs (output0...output15) are programmed is unimportant. However, a minimum of one output must be programmed. If this requirement is not met, a compilation error will occur.

10.0 COMPARE

This function can be used in AutoMax Control Block tasks and UDC Control Block tasks.



Function

Compares INPUT1 against INPUT2
OUTPUT_GTR is set TRUE when INPUT1 is greater than INPUT2
OUTPUT_EQU is set TRUE when INPUT1 is equal to INPUT2
OUTPUT_LES is set TRUE when INPUT1 is less than INPUT2

Program Statement

```
CALL COMPARE(INPUT1=input1%, INPUT2=input2%,          &
    OUTPUT_GTR=greater_than@,                         &
    OUTPUT_EQU=equal@,                                &
    OUTPUT_LES=less_than@)
```

Inputs

I1 (INPUT1) =

Data input 1, type INTEGER. This parameter must be specified.

I2 (INPUT2) =

Data input 2, type INTEGER. This parameter must be specified.

Outputs

GT (OUTPUT_GTR) =

Data output, type BOOLEAN. This is an optional parameter. The output OUTPUT_GTR is set TRUE when INPUT1 is greater then INPUT2.

EQ (OUTPUT_EQU) =

Data output, type BOOLEAN. This is an optional parameter. The output OUTPUT_EQU is set TRUE when INPUT1 is equal to INPUT2.

LT (OUTPUT_LES) =

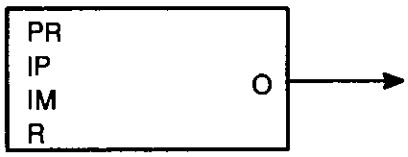
Data output, type BOOLEAN. This is an optional parameter. The output OUTPUT_LES is set TRUE when INPUT1 is less than INPUT2.

Notes

1. The order in which the outputs (OUTPUT_GTR, OUTPUT_EQU, and OUTPUT_LES) are programmed is unimportant. However, a minimum of one output must be programmed. If this requirement is not met, a compilation error will occur.

11.0 COUNTER

This function can be used in AutoMax Control Block tasks and UDC Control Block tasks.



Function

OUTPUT = OUTPUT(n−1) + INPUT(+) − INPUT(−)

Program Statement

```
CALL COUNTER(RESET = reset@,                    &
     PRESET = preset!(%),                       &
     INPUT_PLUS = input_plus%,                  &
     INPUT_MINUS = input_minus%,                &
     OUTPUT = output%)
```

Inputs

PR (PRESET) =

DOUBLE PRECISION INTEGER(!) or INTEGER (%) Counter preset. This parameter must be specified as a variable name (literal value not accepted). The default for this parameter is zero (0). The 32-bit internal counter is preset to this value when RESET is TRUE.

IP (INPUT_PLUS) =

INTEGER plus input. The default for this parameter is zero (0). This input is added to the counter. This parameter can be specified as a variable or a constant.

IM (INPUT_MINUS) =

INTEGER minus input. The default for this parameter is zero (0). This input is subtracted from the counter. This parameter can be specified as a variable or a constant.

R (RESET) =

BOOLEAN counter reset. The default for this parameter is FALSE. This signal causes the counter to be initialized to the preset value.

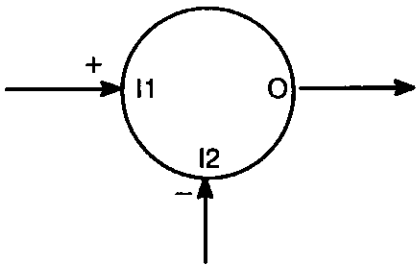
Outputs

O (OUTPUT) =

INTEGER counter output. This parameter must be specified. If the value of the counter exceeds 15 bits plus sign, OUTPUT is limited to the maximum signed value (−32768 to +32767).

12.0 DIFFERENCE

This function can be used in AutoMax Control Block tasks and UDC Control Block tasks.



Function

OUTPUT = INPUT1 – INPUT2

Program Statement

```
CALL DIFFERENCE(INPUT1 = input1%,                    &
     INPUT2 = input2%,                               &
     OUTPUT = output%)
```

Inputs

I1 (INPUT1) =

INTEGER signal input 1. This parameter must be specified.

I2 (INPUT2) =

INTEGER signal input 2. This parameter must be specified.

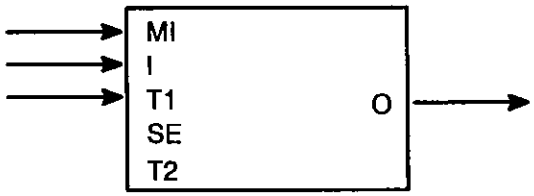
Output

O (OUTPUT) =

INTEGER signal output. This parameter must be specified. OUTPUT is limited to −32768 to +32767. If this limit is exceeded, an error will be logged.

13.0 FUNCTION GENERATOR

This function can be used in AutoMax Control Block tasks and UDC Control Block tasks.



Function

$$\text{subscript.remainder} = \frac{\text{INPUT}*(\text{table\_size} - 1)}{(\text{max\_input} + 1)}$$

OUTPUT = TABLEn%(subscript)+
[(TABLEn%(subscript+1)−TABLEn%(subscript))*
remainder]

OUTPUT is the result of linear interpolation between two points in the table.

Program Statement

```
CALL FUNCTION(INPUT = input%,                    &
    MAX_INPUT = max_input,                       &
    SELECT = select@,                            &
    TABLE1 = table1%,                            &
    TABLE2 = table2%,                            &
    OUTPUT = output%)
```

Inputs

MI (MAX_INPUT) =

The largest possible value of INPUT, type INTEGER. Must be entered explicitly as a numeric literal equal to (2**n)−1 where n is an integer equal to or greater than 1 or equal to or less than 15.

I (INPUT) =

INTEGER signal input. This parameter must be specified. It must be specified as a variable name only (literal value not accepted).

T1 (TABLE) =

INTEGER look up table 1. This parameter must be specified. It must be specified as a variable name only (literal value not accepted). It must be defined as an array with the size (number of points) equal to (2**m)+1 where m is an integer less than or equal to n for MAX_INPUT. This array definition occurs when defining the variable using a LOCAL or COMMON statement in the task [LOCAL TABLE1 %(16)]. Values must be calculated and stored for all points in the array.

LOCAL TABLE1%(16), for example, will index into the array beginning with the first element, which has the subscript zero(0), not 1. Therefore, to define an array having the number of elements of $2^{**m}+1,2^{**m}$ is used when defining the array. To define the common array "FUNCTION_TBL" with 17 elements ($2^4 + 1$) and a local array "TABLE1%" with 9 elements ($2^3 + 1$), the definition statements would be:

```
10 COMMON FUNCTION_TBL%(16)
20 LOCAL TABLE1%(8)
```

SE (SELECT) =

BOOLEAN TABLE. The default for the parameter is FALSE. When FALSE, TABLE1 will be selected; otherwise, TABLE2 will be selected.

T2 (TABLE2) =

INTEGER look up table 2. This parameter is optional. If specified, it must be as a variable name only (literal value not accepted). The same table definition specifications as for TABLE1 apply. If TABLE2 is specified, SELECT should also be specified, and vice versa. If specified, TABLE2 must be the same size as TABLE 1.

Outputs

O (OUTPUT) =

INTEGER signal output. This parameter must be specified. If SELECT is FALSE, OUTPUT will be obtained from TABLE1; otherwise, OUTPUT will be obtained from TABLE2. OUTPUT is linearly interpolated between points in the table.

If INPUT goes negative, the computed value for output will be negated as well. This will have the effect of folding the function defined in the table from the first quadrant to the third quadrant (assuming the table entries are in the first quadrant).

14.0 INVERTER

This function can be used in AutoMax Control Block tasks and UDC Control Block tasks.

Function

If ENABLE = TRUE then
OUTPUT = −INPUT

else

OUTPUT = INPUT

Program Statement

```
CALL INVERTER (ENABLE = enable@, INPUT = input%,          &
    OUTPUT = output%)
```

Inputs

(EN) ENABLE =

Enable input, type BOOLEAN. This is an optional parameter. The default is TRUE. When ENABLE is TRUE, OUTPUT is set equal to the complemented value of INPUT. When ENABLE is FALSE, OUTPUT is set equal to the value of INPUT.

I (INPUT) =

INTEGER signal input. This parameter must be specified. It must be specified as a variable name only (literal value not accepted).

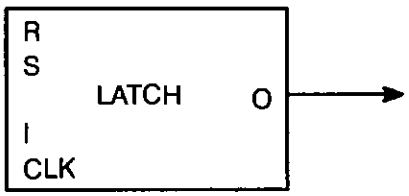
Outputs

O (OUTPUT) =

INTEGER signal output. This parameter must be specified. When ENABLE is TRUE, OUTPUT = −INPUT. When ENABLE is FALSE, OUTPUT = INPUT.

15.0 LATCH

This function can be used in AutoMax Control Block tasks and UDC Control Block tasks.



Function

If RESET is TRUE set OUTPUT FALSE

else

if RESET is FALSE and SET is TRUE
set OUTPUT TRUE

else if

RESET and SET are FALSE and CLOCK is TRUE
set OUTPUT to the state of INPUT

else

OUTPUT state is unchanged.

Program Statement

```
CALL LATCH (RESET=reset@, SET=set@,                    &
    CLOCK=clock@, INPUT=input@,                        &
    OUTPUT=output@)
```

Inputs

R (RESET) =

Reset input, type BOOLEAN. This is an optional parameter. The default is FALSE. When RESET is TRUE the OUTPUT will be set FALSE.

S (SET) =

Set input, type BOOLEAN. This is an optional parameter. The default is FALSE. When RESET is FALSE and SET is TRUE the OUTPUT will be set TRUE.

CLK (CLOCK) =

Clock input, type BOOLEAN. This is an optional parameter. The default is FALSE. When RESET and SET are FALSE and CLOCK is TRUE, set OUTPUT equal to the state of INPUT.

I (INPUT) =

Data input, type BOOLEAN. This is an optional parameter. The default is FALSE. Determines the state of OUTPUT if RESET and SET are FALSE and CLOCK is TRUE.

Outputs

O (OUTPUT) =

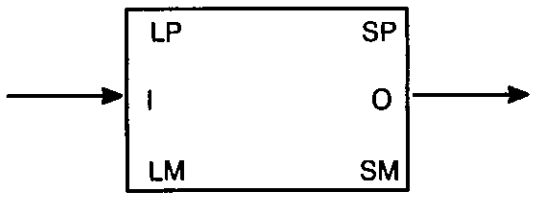
Data output, type BOOLEAN. This parameter must be specified. The state of OUTPUT is determined by the state of the inputs RESET, SET, CLOCK, and INPUT. See the functional description above.

Notes

1. The order in which the input parameters (RESET, SET, CLOCK, and INPUT) are programmed is unimportant. However, a minimum of two of the four input parameters must be programmed. If this requirement is not met, a compilation error will occur.

16.0 LIMIT

This function can be used in AutoMax Control Block tasks and UDC Control Block tasks.



Function

OUTPUT = INPUT within the range LIMIT(+) TO LIMIT(−). If INPUT exceeds either limit, OUTPUT is held at that limit and the proper SATURATED output will be set.

Program Statement

```
CALL LIMIT(INPUT = input%,                              &
    LIMIT_PLUS = limit_plus%,                           &
    LIMIT_MINUS = limit_minus%,                         &
    SATURATED_PLUS = saturated_plus@,                   &
    SATURATED_MINUS = saturated_minus@,                 &
    OUTPUT = output%)
```

Inputs

LP (LIMIT_PLUS) =

INTEGER upper limit. The default for this parameter is +32767. This parameter will limit OUTPUT from becoming more positive. It will not prevent the output value from becoming more negative.

I (INPUT) =

INTEGER signal input. This parameter must be specified. It must be specified as a variable name only (literal value not accepted).

LM (LIMIT_MINUS) =

INTEGER lower limit. The default for this parameter is −32768. This parameter will limit OUTPUT from becoming more negative. It will not prevent the output value from becoming more positive.

Outputs

SP (SATURATED_PLUS) =

BOOLEAN SATURATED plus output. This parameter is optional. TRUE if OUTPUT reaches LIMIT_PLUS.

O (OUTPUT) =

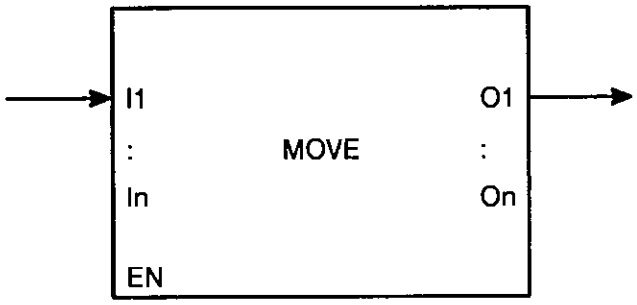
INTEGER signal output. This parameter must be specified.

SM (SATURATED_MINUS) =

BOOLEAN saturated minus output. This parameter is optional. TRUE if OUTPUT reaches LIMIT_MINUS.

17.0 MOVE

This function can be used in AutoMax Control Block tasks and UDC Control Block tasks.



Maximum 'n' = 8

Function

If ENABLE = TRUE then
  OUTPUT1 = INPUT1
  : :
  OUTPUTn = INPUTn

Program Statement

```
CALL MOVE (ENABLE=enable@,                                    &
    INPUT1=input1%, OUTPUT1=output1%, ...                     &
    INPUTn=inputn%, OUTPUTn=outputn% )
```

Inputs

EN (ENABLE) =

Enable input, type BOOLEAN. This is an optional parameter. The default is TRUE. When ENABLE is TRUE the values read at the input(s) are transferred to the corresponding outputs. When ENABLE is FALSE, the outputs are not updated, i.e., the values will remain unchanged from the previous scan.

I1 (INPUT1) =

Signal input 1, type INTEGER. This parameter must be specified.

In (INPUTn) =

Signal input n, type INTEGER. This is an optional parameter. A maximum of 8 inputs can be specified. Unused inputs are ignored. Each specified input must have a corresponding output.

Outputs

O1 (OUTPUT1) =

Data output 1, type INTEGER. This parameter must be specified.

On (OUTPUTn) =

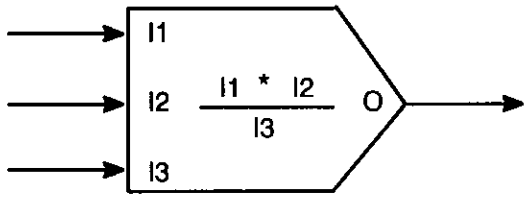
Data output n, type INTEGER. This is an optional parameter. The number of outputs specified must equal the number of inputs.

Notes

1. The order in which the input/output pairs are entered is unimportant. However, for every INPUT(n) programmed, an OUTPUT(n) must also be programmed. In addition, all input/output pairs must be contiguous beginning with input/output pair 1. If these requirements are not met, a compilation error will occur.
2. When ENABLE is FALSE, the outputs are not updated. Therefore if an output is forced and then unforced it will not return to its original value.

18.0 MULTIPLY AND DIVIDE

This function can be used in AutoMax Control Block tasks and UDC Control Block tasks.



Function

OUTPUT = INPUT1 * INPUT2 ÷ INPUT3

Program Statement

```
CALL MULTIPLY_DIVIDE(INPUT1 = input1%,                    &
    INPUT2 = input2%,                                     &
    INPUT3 = input3%,                                     &
    OUTPUT = output%)
```

Inputs

I1 (INPUT1) =

INTEGER signal input 1. This parameter must be specified.

I2 (INPUT2) =

INTEGER signal input 2. This parameter must be specified.

I3 (INPUT3) =

INTEGER signal input 3. This parameter must be specified.

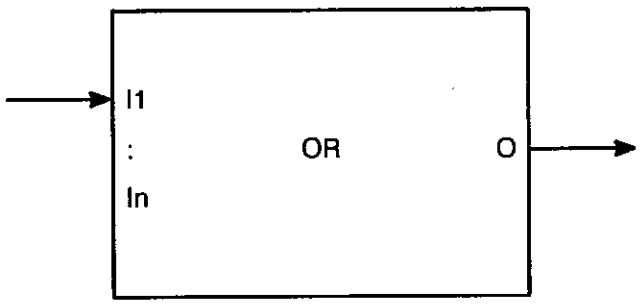
Outputs

O (OUTPUT) =

INTEGER signal output. This parameter must be specified. If the result exceeds the range −32768 to +32767, the result will be clamped to −32768 or +32767, producing OUTPUT.

19.0 OR

This function can be used in AutoMax Control Block tasks and UDC Control Block tasks.



Maximum 'n' = 8

Function

OUTPUT = INPUT1 or ... INPUTn

Program Statement

```
CALL OR  (INPUT1=input1@,                    &
    INPUTn=inputn@,                          &
    OUTPUT=output@)
```

Inputs

I1 (INPUT1) =

Signal input 1, type BOOLEAN. This parameter must be specified.

In (INPUTn) =

Signal input n, type BOOLEAN. This is an optional parameter. A maximum of 8 inputs can be specified.

Outputs

O (OUTPUT) =

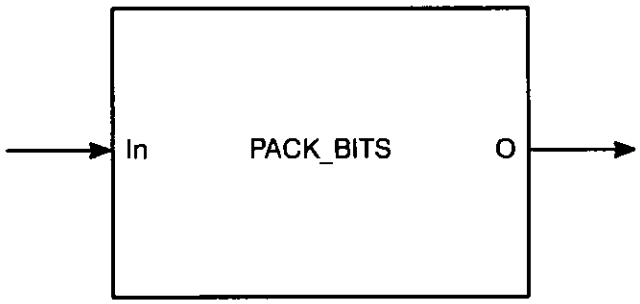
Data output, type BOOLEAN. This parameter must be specified. If any of the specified inputs are TRUE then OUTPUT will be TRUE.

Notes

1. The inputs programmed must be contiguous beginning with input 1. A compilation error will occur if this requirement is not met.

20.0 PACK BITS

This function can be used in AutoMax Control Block tasks and UDC Control Block tasks.



'n' = 0...15

Function

BITn in OUTPUT is set to the state of INPUTn. If INPUTn is not programmed, then BITn in OUTPUT is set FALSE.

Program Statement

```
CALL PACK_BITS(INPUT0=input0@, ... INPUTn=inputn@,          &
    OUTPUT=output%)
```

Inputs

In (INPUTn) =

Signal input 0...15, type BOOLEAN. The inputs can be specified in any order.

Outputs

O (OUTPUT) =

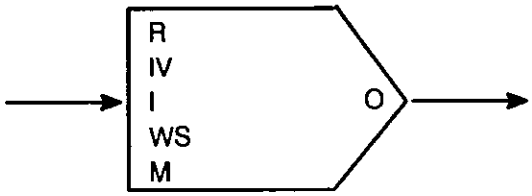
Data output, type INTEGER. This parameter must be specified.

Notes

1. The order in which the inputs (input0...input15) are programmed is unimportant. However, a minimum of one input must be programmed. If this requirement is not met, a compilation error will occur.

21.0 PULSE MULTIPLIER

This function can be used in AutoMax Control Block tasks and UDC Control Block tasks.



Function

If WORD_SIZE > 0 (Relative mode), then

    If RESET@ = TRUE then
        ERROR = 0
        Rem (n-1) = 0
        Error = sign extended (Error masked with
            (16-WORD_SIZE))

    else if TRUE to FALSE transition on RESET@ then
        Input (n-1) = INITIAL VALUE
        Error = INPUT - Input (n-1)
        Error = sign extended (Error masked with
            (16-WORD_SIZE))

    else (RESET@ = all other conditions)
        if (FIRST PASS) then
            INPUT (n-1) = INPUT
        Error = INPUT - INPUT (n-1)
        Error = sign extended (Error masked with
            (16-WORD_SIZE))

else (Absolute mode) then

    If RESET@ = TRUE then
        Error = 0
        Rem (n-1) = 0

    else (RESET@ = all other conditions)
        Error = INPUT

(Calculate OUTPUT and Remainder (n-1) for both modes)
Ans (real) = ((Error * MULTIPLIER) / 32768)+Rem (n-1))
OUTPUT = INT (Ans)
Rem (n-1) = Ans - OUTPUT

Program Statement

```
CALL PULSE_MULT(INPUT = input%,                    &
    RESET = reset@,                                &
    WORD_SIZE = nnn,                               &
    INITIAL_VALUE = initial_value%,                &
    MULTIPLIER = multiplier%,                      &
    OUTPUT = output%)
```

Inputs

R (RESET) =

BOOLEAN multiplier reset. The default for this parameter is FALSE. If TRUE, OUTPUT is held to zero and all internal registers are set to zero. If TRUE to FALSE transition, the INPUT(n-1)term will be obtained from INITIAL_VALUE [not applicable when used in absolute mode (WORD_SIZE = 0)]. If FALSE, the block will function normally.

IV (INITIAL_VALUE) =

INTEGER preset of INPUT (n-1). The default for this parameter is zero. On the TRUE to FALSE transition of RESET, INITIAL_VALUE is used as INPUT (n-1) when computing relative counts/scan. INITIAL_VALUE must be right-justified within the 16-bit word regardless of the WORD_SIZE value.

I (INPUT) =

INTEGER signal input. This parameter must be specified. It must be specified as a variable name only (literal value not accepted).

WS (WORD_SIZE) =

INTEGER word size of input (in bits). Specifies the number of significant bits in INPUT beginning with bit 0 (right-justified). The default for this parameter is zero. If specified, it must be entered explicitly as a numeric literal between 0 and 16, inclusive. If WORD_SIZE equals 0, INPUT must be in counts/scan (absolute mode). If WORD_SIZE does not equal 0, INPUT must be right-justified and a relative counts/scan value is computed as [(INPUT - INPUT(n-1)].

M (MULTIPLIER) =

INTEGER multiplier ratio. The default for this parameter is 32768, giving an overall gain of 1.0. This parameter divided by 32768 controls the ratio of input to output counts.

Outputs

O (OUTPUT) =

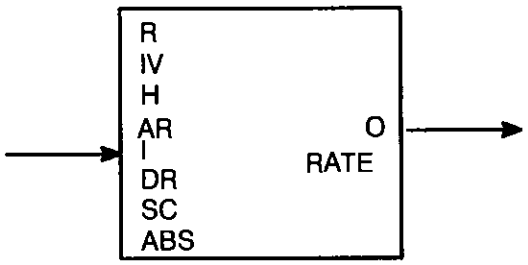
INTEGER signal output. This parameter must be specified.

Notes

1. Not programming (defaulting) the MULTIPLIER parameter is the only way of achieving a gain equal to 1. If programmed, the nearest value to 1.0 that the overall gain can achieve is 32767 ÷ 32768, or 0.9999695.
2. The intended use of this block is to provide an interface from a frequency input module, such as the resolver feedback, to the digital system. It also provides a means to scale position information properly without losing any information from one scan to the next. This is accomplished by saving the remainder after the divide operation and adding it during the next scan calculations.

22.0 RAMP

This function can be used in AutoMax Control Block tasks and UDC Control Block tasks.



Function

For a change in INPUT, OUTPUT will ramp toward the new INPUT value. During steady state operation, OUTPUT = INPUT.

Two types of RAMP generators are provided: a normal (algebraic) ramp and an absolute value ramp. For a normal ramp, the accel condition is defined by an input that is becoming more positive, and a decel condition is defined by an input that is becoming more negative. For an absolute value ramp, the accel condition is defined by an input moving away from zero, and a decel condition is defined by an input moving towards zero.

Program Statement

```
CALL RAMP(INPUT = input%,                    &
    ABS_RAMP = TRUE/FALSE,                   &
    RESET = reset@,                          &
    INITIAL_VALUE = initial_value%,          &
    HOLD = hold@,                            &
    ACCEL_RATE = accel_rate%,                &
    DECEL_RATE = decel_rate%,                &
    SCALE = nnnnn,                           &
    OUTPUT = output%,                        &
    RATE = rate%)
```

Inputs

R (RESET) =

BOOLEAN ramp reset. The default for this parameter is FALSE. This parameter will hold OUTPUT to INITIAL_VALUE when TRUE. The internal ramp register is also reset.

IV (INITIAL_VALUE) =

INTEGER initial ramp value. The default for this parameter is zero. When RESET = TRUE, OUTPUT will equal INITIAL_VALUE.

H (HOLD) =

BOOLEAN ramp hold. The default for this parameter is FALSE. This parameter will hold OUTPUT at its current value. OUTPUT will continue to ramp from that value when HOLD is FALSE.

AR (ACCEL_RATE) =

INTEGER acceleration rate (in units of counts per scan). The absolute value of this input is performed to obtain the acceleration rate. The default for this parameter is 32767.

I (INPUT) =

INTEGER signal input. This parameter must be specified.

DR (DECEL_RATE) =

INTEGER deceleration rate (in units of counts per scan). The absolute value of this input is performed to obtain the deceleration rate. The default for this parameter is 32767.

SC (SCALE) =

INTEGER scale factor for both ACCEL_RATE and DECEL_RATE. The default for this parameter is 1. This parameter must be entered explicitly as a numeric literal and, therefore, cannot be modified while the task is active. This parameter must be positive between 1 and 32767 inclusive.

ABS (ABS_RAMP) =

BOOLEAN ramp type. The default for this parameter is TRUE. If programmed, it must be entered explicitly as a boolean literal and, therefore, cannot be modified while the task is active. If TRUE, the block will function as an absolute value ramp; otherwise, it will function as a normal (algebraic) ramp.

Outputs

O (OUTPUT) =

INTEGER signal output. This parameter must be specified.

RATE (RATE) =

INTEGER internal change in the OUTPUT * SCALE. This parameter is optional. It is computed from the internal 32-bit value as (OUTPUT * SCALE) - (OUTPUT(n-1) * SCALE).

Notes

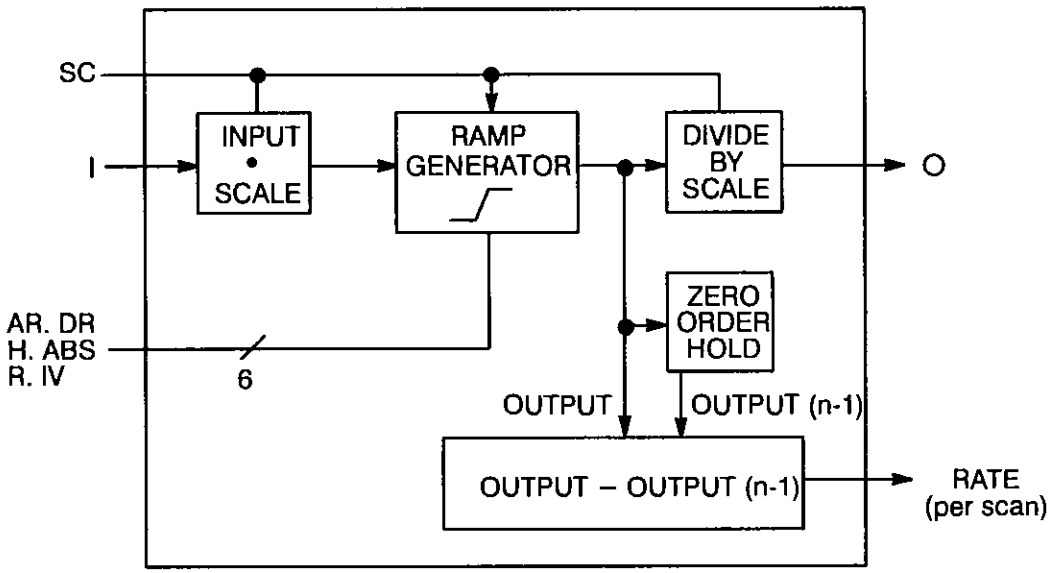
The SCALE input is optional. It is used only to change the units for the two rate inputs (ACCEL_RATE and DECEL_RATE) from per-scan to per-second units. To do this, the SCALE input is set to the number of times the block is executed in one second.

For example, if the RAMP block is used in a UDC task running at 20 ticks, SCALE should be set to 100. On the UDC, 1 tick=0.0005 seconds, so 20 ticks=0.01 seconds. SCALE is then (1/0.01)=100.

In this example, if an accel rate of 500 counts per second is desired, ACCEL_RATE would be set to 500. If the SCALE input was not used

(it defaults to 1), ACCEL_RATE would have to be set to 5 to achieve the desired 500 counts per second.

RAMP Internal Block Diagram



1. When accel rate and/or decel rate is equal to zero (0), the ramp will effectively be held (not permitted to move) in that direction.

2. The actual rate of change of the output is inversely proportional to the scan time. Thus, the actual rates in counts/second can be calculated as:

$$\text{RATE} = \frac{\text{rate \%}}{\text{scale\%}} * \frac{1}{Ts}$$

where:

rate% = accel_rate% or decel_rate%
Ts = scan period in seconds/scan

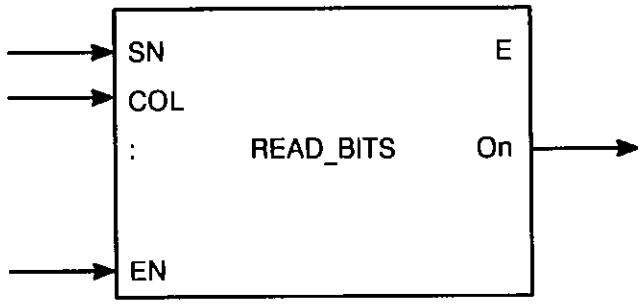
The slowest rate, therefore, is when rate% is equal to its minimum value (1) and scale% is equal to its maximum value (32767). Similarly, the fastest rate is when rate% is equal to its maximum value (32767) and scale% is equal to its minimum value (1).

For example, when T_s = 5.5 msec:

Min rate = (1 ÷ 32767) ÷ .0055 = .00554882 counts/sec
Max rate = (32767 ÷ 1) ÷ .0055 = 5957636.4 counts/sec

23.0 READ BITS

This function can be used in AutoMax Control Block tasks only. It cannot be used in UDC Control Block tasks.



Maximum 'n' = 8

Function

This function reads data from a column in the specified BOOLEAN data structure.

Program Statement

```
CALL READ_BITS(STRUCTURE_NAME=structure_name@,          &
    COLUMN=column%, ENABLE=enable@,                     &
    ERROR=error@,                                       &
    OUTPUT1=output1%,...OUTPUTn=outputn%)
```

Inputs

SN (STRUCTURE_NAME) =

Name of the BOOLEAN data structure where data is to be read from. This parameter must be specified by name only (literal value not accepted). The data structure name is limited to a maximum length of 15 characters and must be type BOOLEAN. The specified data structure must be created by a control block within the task. Refer to the SHIFT_BITS block for an example of a control block that creates a BOOLEAN data structure.

COL (COLUMN) =

Selects a column within the specified BOOLEAN data structure, type INTEGER. This parameter is required. The columns are numbered from 0 to MCOL - 1, where MCOL is equal to the number of columns (depth) that were defined by the control block that created the data structure.

EN (ENABLE) =

Enable input, type BOOLEAN. This parameter is required. The state(s) read from the column specified by COLUMN are written to the output(s) when ENABLE is TRUE. If this parameter is FALSE, the state(s) at the outputs will remain unchanged from the previous scan.

Outputs

E (ERROR) =

Error output, type BOOLEAN. This is an optional parameter. The output is TRUE if the value of COLUMN selects a non-existent column for the specified data structure. Valid values for COLUMN range from 0 to MCOL - 1. See COLUMN above.

On (OUTPUTn) =

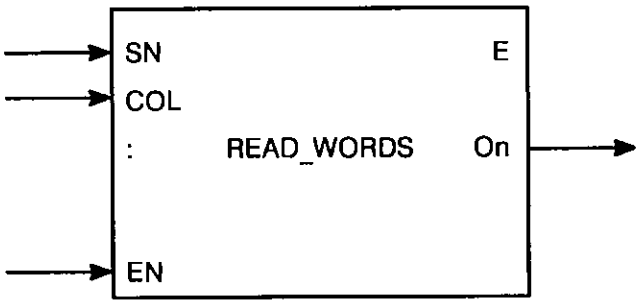
Data output n, type BOOLEAN. The outputs can be specified in any order.

Notes

1. The READ_BITS block must reference a BOOLEAN data structure that was created by a control block within this task. A minimum of one output must be programmed. The order in which the outputs (output1...output8) are programmed is not important. However, all of the outputs programmed by the READ_BITS block must also be defined by the control block that created the data structure. If these requirements are not met, a compilation error will occur.
2. If the value of COLUMN selects a non-existent column, the output ERROR is set TRUE, the output (s) of the READ_BITS block are set FALSE, and the appropriate run time error is logged.
3. When ENABLE is FALSE, the output(s) are not updated. Therefore if an output is forced and then unforced, it will not return to its original value.

24.0 READ WORDS

This function can be used in AutoMax Control Block tasks only. It cannot be used in UDC Control Block tasks.



Maximum 'n' = 8

Function

This function reads data from a column in the specified INTEGER data structure.

Program Statement

```
CALL READ_WORDS(STRUCTURE_NAME=structure_name%,     &
    COLUMN=column%, ENABLE=enable@,                 &
    ERROR=error@,                                   &
    OUTPUT1=output1%,...OUTPUTn=outputn%)
```

Inputs

SN (STRUCTURE_NAME) =

Name of the INTEGER data structure where data is to be read from. This parameter must be specified by name only (literal value not accepted). The data structure name is limited to a maximum length of 15 characters and must be type INTEGER. The specified data structure must be created by a control block within the task. Refer to the SHIFT_WORDS block for an example of a control block that creates an INTEGER data structure.

COL (COLUMN) =

Selects a column within the specified INTEGER data structure, type INTEGER. This parameter is required. The columns are numbered from 0 to MCOL - 1, where MCOL is equal to the number of columns (depth) defined by the control block that created the data structure.

EN (ENABLE) =

Enable input, type BOOLEAN. This parameter is required. The values read from the column specified by COLUMN are written to the outputs when ENABLE is TRUE. If this parameter is FALSE, the values at the outputs will remain unchanged from the previous scan.

Outputs

E (ERROR) =

Error output, type BOOLEAN. This is an optional parameter. The output is TRUE if the value of COLUMN selects a non-existent column for the specified data structure. Valid values for COLUMN range from 0 to MCOL - 1; see COLUMN above.

On (OUTPUTn) =

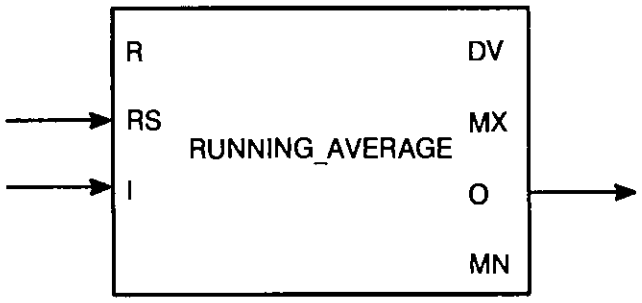
Data output n, type INTEGER. The outputs can be specified in any order.

Notes

1. The READ_WORDS block must reference an INTEGER data structure that was created by a control block within the task. A minimum of one output must be programmed. The order in which the outputs (output1...output8) are programmed is unimportant. However, all of the outputs programmed by the READ_WORDS block must also be defined by the control block that created the data structure. If these requirements are not met, a compilation error will occur.
2. If the value of COLUMN selects a non-existent column, the output ERROR is set TRUE, the output(s) of the READ_WORDS block are set equal to zero, and the appropriate run time error is logged.
3. When ENABLE if FALSE, the output(s) are not updated. Therefore if an output is forced and then unforced, it will not return to its original value.

25.0 RUNNING AVERAGE

This function can be used in AutoMax Control Block tasks and UDC Control Block tasks.



Function

OUTPUT = $\frac{\text{(INPUT(n) + ...INPUT(n+1))}}{\text{(REQUIRED\_SAMPLES)}}$

The OUTPUT is updated each scan with the average of the samples read for INPUT over the last RS scans.

Program Statement

```
CALL RUNNING_AVERAGE(REQUIRED_SAMPLES=req_sam,      &
    RESET=reset@,                                   &
    INPUT=input%,                                   &
    DATA_VALID=data_valid@,                         &
    MAX_VALUE=max_value%, MIN_VALUE=min_value%,     &
    OUTPUT=output%)
```

Inputs

RS (REQUIRED_SAMPLES) =

Required number of samples to average, type INTEGER. This parameter must be entered explicitly as a numeric literal. The value entered for REQUIRED_SAMPLES can range from 1 to 32767. See note 1 for details.

R (RESET) =

Reset input, type BOOLEAN. This is an optional parameter. The default is FALSE. When this input is true, the following occur: the internal storage areas are initialized, OUTPUT is set to zero, VALID_DATA is set FALSE, MAX_VALUE is set to -32768, and MIN_VALUE is set to 32767.

I (INPUT) =

Signal input, type INTEGER. This parameter must be specified as a numeric symbol only (literal value not accepted).

Outputs

DV (DATA_VALID) =

Data valid output, type BOOLEAN. This is an optional parameter. DATA_VALID is set TRUE after a minimum of RS samples have been read since RESET went FALSE. When RESET is TRUE, DATA_VALID is set FALSE.

MX (MAX_VALUE) =

Maximum value output, type INTEGER. This is an optional parameter. This parameter outputs the maximum value sampled for INPUT since RESET went FALSE. When RESET is TRUE, MAX_VALUE is set to -32768.

MN (MIN_VALUE) =

Minimum value output, type INTEGER. This is an optional parameter. This parameter outputs the minimum value sampled for INPUT since RESET went FALSE. When RESET is TRUE, MIN_VALUE is set to 32767.

O (OUTPUT) =

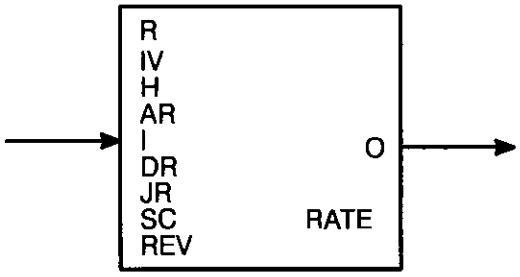
Data output, type INTEGER. This parameter must be specified. OUTPUT is updated each scan with the average of the RS samples read for INPUT over the last RS scan. When RESET is TRUE, OUTPUT is set to a value of zero.

Notes

1. The RUNNING_AVERAGE block creates an internal data storage area that is used to store the last RS samples read for INPUT. The size of the internal data storage area in bytes is equal to two times the required number of samples. If the size of the internal data storage area exceeds 32767 bytes, a compilation error will occur. In addition, since the internal data storage area is local to the control block task, its size is further limited by the maximum size of the total data storage area that the task can allocate for all the local variables.

26.0 S CURVE

This function can be used in AutoMax Control Block tasks and UDC Control Block tasks



Function

The S_CURVE block performs the same basic function as the RAMP block with a jerk rate added. The jerk rate is the maximum rate of change of the rate used to ramp output to input, just as the accel and decel rates are the maximum rate of change of output to input. It is, therefore, the second derivative of the input. The primary requirement of the S_CURVE function is to ensure that the rate never changes by more than the specified jerk rate.

When a step change occurs on the input, the rate increases at the jerk rate up to the accel or decel rate. The accel or decel rate is maintained until output reaches a distance from input at which point the rate must begin decreasing. Rate will decrease by the jerk rate such that output equals input when rate is less than or equal to the jerk rate. In some cases, depending on the values of accel, decel, and jerk, the accel or decel rate may not be reached before the rate must begin decreasing by the jerk rate.

If the reverse bit = TRUE, the input accel rate will become the decel rate and the input decel rate will become the accel rate. This provides a function similar to the "motor" type RAMP block (ABS_RAMP = TRUE). With the S_CURVE block, however, this function can be dynamically controlled by the application.

Program Statement

```
CALL S_CURVE(INPUT = input%,                    &
    RESET = reset@,                             &
    HOLD = hold@,                               &
    ACCEL_RATE = accel_rate%,                   &
    DECEL_RATE = decel_rate%,                   &
    JERK_RATE = jerk_rate%,                     &
    SCALE = nnnnn,                              &
    REVERSE = reverse@,                         &
    INITIAL_VALUE = initial_value%,             &
    OUTPUT = output%,                           &
    RATE = rate%)
```

Inputs

R (RESET) =

BOOLEAN S_CURVE reset. The default for this parameter is FALSE. When TRUE, OUTPUT will be held equal to INITIAL_VALUE.

IV (INITIAL_VALUE) =

INTEGER initial value of S_CURVE.The default for this parameter is zero. When RESET = TRUE, OUTPUT will equal INITIAL_VALUE.

H (HOLD) =

BOOLEAN S_CURVE hold. The default for this parameter is FALSE. When TRUE, OUTPUT will be held at its current value. (See item 3 under "S_CURVE Internal Block Diagram.") OUTPUT will continue to move from that value until it equals INPUT when HOLD is FALSE.

AR (ACCEL_RATE) =

INTEGER acceleration rate (in units of counts per scan). The absolute value of this input is used to obtain the maximum acceleration rate from OUTPUT to INPUT. The default for this parameter is 32767.

I (INPUT) =

INTEGER signal input (initial output in equation). This parameter must be specified.

DR (DECEL_RATE) =

INTEGER deceleration rate (in units of counts per scan). The absolute value of this input is used to obtain the maximum deceleration rate from OUTPUT to INPUT. The default for this parameter is 32767.

JR (JERK_RATE) =

INTEGER jerk rate (in units of counts per scan^2). The absolute value of this input is used to obtain the jerk rate. The default for this parameter is 32767. The jerk rate defines the rate of change in the rate used when ramping output to input.

SC (SCALE) =

INTEGER scale factor forACCEL_RATE, DECEL_RATE and JERK_RATE. The default for this parameter is 1. This parameter must be entered explicitly as a numeric literal and, therefore, cannot be modified while the task is active. This parameter must be positive between 1 and 32767 inclusive.

REV (REVERSE) =

BOOLEAN S_CURVE direction. The default for this parameter is FALSE. If TRUE, the ACCEL_RATE will be used as the deceleration rate and the DECEL_RATE will be used as the acceleration rate.

Outputs

O (OUTPUT) =

INTEGER signal output (final output in equations). This parameter must be specified.

RATE (RATE) =

INTEGER. Internal change in the OUTPUT * SCALE. This parameter is optional. It is calculated from the internal 32-bit value of OUTPUT * SCALE.

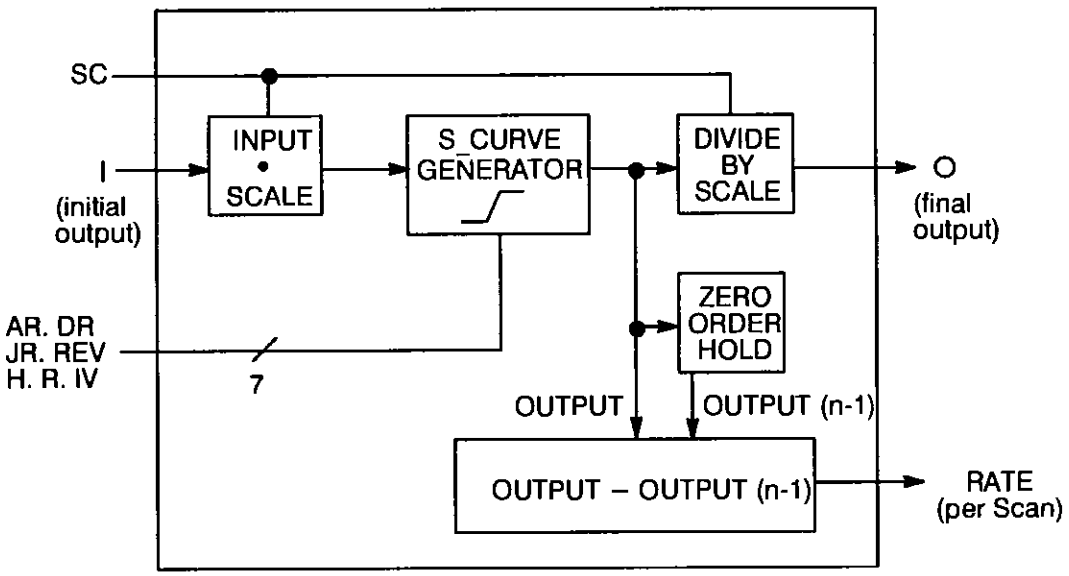
Notes

The SCALE input is optional. It is used only to change the units for the two rate inputs (ACCEL_RATE and DECEL_RATE) from per-scan to per-second units. To do this, the SCALE input is set to the number of times the block is executed in one second.

For example, if the S_CURVE block is used in a UDC task running at 20 ticks, SCALE should be set to 100. On the UDC, 1 tick=0.0005 seconds, so 20 ticks=0.01 seconds. SCALE is then (1/0.01)=100.

In this example, if an accel rate of 500 counts per second is desired, ACCEL_RATE would be set to 500. If the SCALE input was not used (it defaults to 1), ACCEL_RATE would have to be set to 5 to achieve the desired 500 counts per second.

S_CURVE Internal Block Diagram



1. When accel rate and/or decel rate is equal to zero (0), the block will effectively be held (not permitted to move) in that direction.
2. The actual rate of change of the output is inversely proportional to the scan time. Thus, the actual rates in counts/second can be calculated as:

$$RATE = \frac{rate\ \%}{scale\%} * \frac{1}{Ts}$$

where:

rate% = accel_rate% or decel_rate% (or jerk_rate% ÷ Ts)
Ts = scan period in seconds/scan

The slowest rate, therefore, is when rate% is equal to its minimum (1) and scale% is equal to its maximum value (32767). Similarly, the fastest rate is when rate% is equal to its maximum value (32767) and scale% is equal to its minimum value (1).

For example, when Ts = 5.5 msec:

Min rate = (1 ÷ 32767) ÷ .0055 = .00554882 counts/sec
Max rate = (32767 ÷ 1) ÷ .0055 = 5957636.4 counts/sec

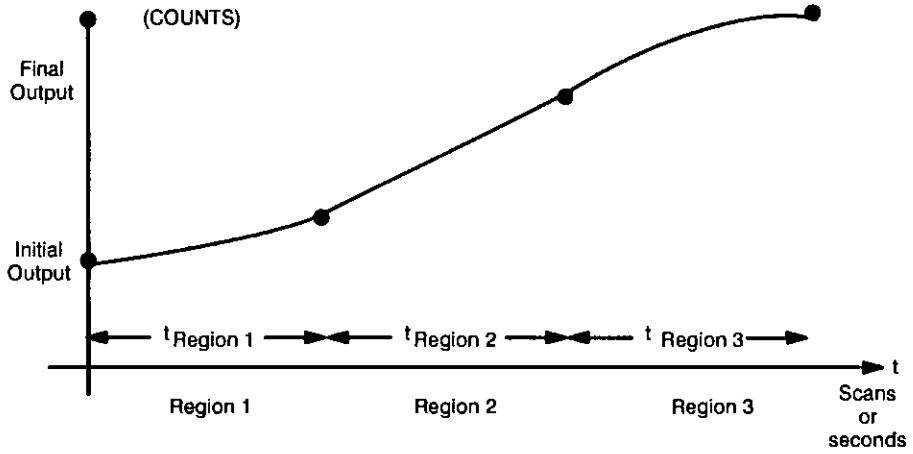
3. When OUTPUT (final) is moving toward input and HOLD is SET = TRUE, the rate will immediately begin decreasing towards zero at the jerk rate. OUTPUT (initial) will be held at whatever value it had when the rate reached zero. This is due to the jerk rate requirement. Therefore, when OUTPUT is finally held constant, it will have a value different from its value the instant that HOLD was SET = TRUE.

4. If JERK_RATE is greater than or equal to ACCEL_RATE and DECEL_RATE, the function will be identical to that of the RAMP block.

5. Reducing the JERK_RATE during a transition may cause the following to occur: the OUTPUT may overshoot the INPUT and/or the calculated output value may exceed the limits of an integer variable (±32767). If overshoot occurs, it is the result of enforcing the entered JERK_RATE. To enforce the JERK_RATE, if the RATE is at a value which, when the JERK_RATE is lowered, cannot be decreased to zero by the new JERK_RATE before OUTPUT = INPUT, OUTPUT must overshoot INPUT. This can be avoided by decreasing JERK_RATE in small steps while tuning or only decreasing JERK_RATE while OUTPUT = INPUT (not during a transition). If the computed output value exceeds +32767 or −32768, the OUTPUT will be clamped to +32767 or −32768, respectively, and an error will be logged. Also, the internal value for OUTPUT will be clamped to avoid internal overshoot and the RATE will be set to zero.

6. The time that is required for the output to equal a change in the input is a function of ACCEL_RATE, JERK_RATE, and the difference between INPUT and OUTPUT.

System Reaches ACCEL_RATE



$$tTotal = \frac{FINAL\ OUTPUT - INITIAL\ OUTPUT}{ACCEL} + \frac{ACCEL}{JERK}$$

Region 1 Equations

$$t_1 = \frac{ACCEL}{JERK}$$

$$y\ (t) = INITIAL\ OUTPUT + {}^1/_2\ (JERK)\ t^2$$

Region 2 Equations

$$t_2 = \frac{JERK\left(FINAL\ OUTPUT - INITIAL\ OUTPUT\right) - ACCEL^2}{JERK * ACCEL}$$

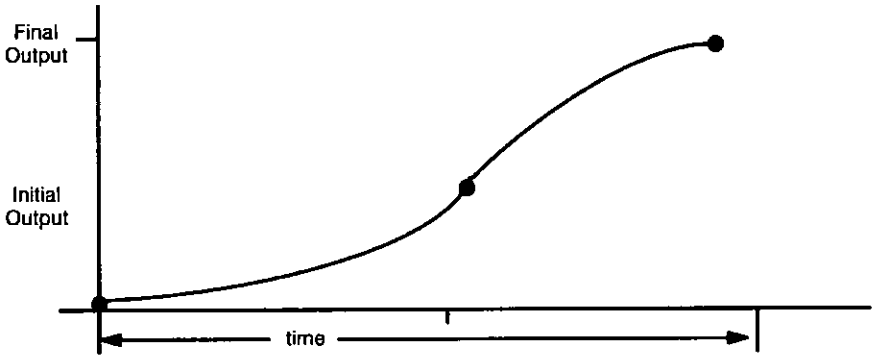
$$y\ (t) = INITIAL\ OUTPUT + (ACCEL)\ t - \left(\frac{ACCEL^2}{2 * JERK}\right)$$

Region 3 Equations

$$t_3 = \frac{ACCEL}{JERK}$$

$$y\ (t) = FINAL\ OUTPUT - {}^1/_2\ (JERK) * \left[t - \frac{FINAL\ OUTPUT - INITIAL\ OUTPUT}{ACCEL} - \frac{ACCEL}{JERK}\right]^2$$

System Does Not Reach ACCEL_RATE

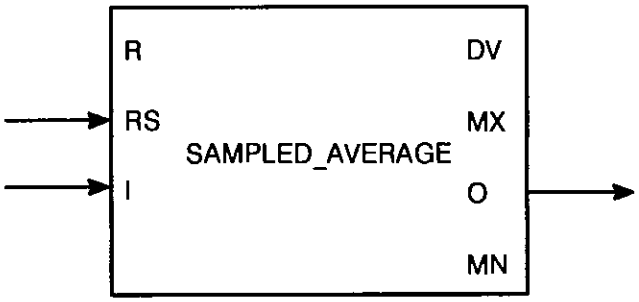


When INITIAL OUTPUT − FINAL OUTPUT < (ACCEL_RATE^2/JERK_RATE)

$$\text{time} = 2^* \sqrt{\frac{\text{INITIAL OUTPUT} - \text{FINAL OUTPUT}}{\text{JERK\_RATE}}}$$

27.0 SAMPLED AVERAGE

This function can be used in AutoMax Control Block tasks and UDC Control Block tasks.



Function

$$OUTPUT = \frac{(INPUT(1) + ...INPUT(RS))}{(REQUIRED\_SAMPLES)}$$

The OUTPUT is updated once every RS scans with the average of the samples read for INPUT during the last RS scans.

Program Statement

```
CALL SAMPLED_AVERAGE(REQUIRED_SAMPLES=req_sam,        &
    INPUT=input%,                                     &
    RESET=reset@,                                     &
    DATA_VALID=data_valid@,                           &
    MAX_VALUE=max_value%, MIN_VALUE=min_value%,       &
    OUTPUT=output%)
```

Inputs

RS (REQUIRED_SAMPLES) =

Required number of samples to average, type INTEGER. This parameter must be entered explicitly as a numeric literal or symbol. The value entered for REQUIRED_SAMPLES can range from 1 to 32767. Values that are ≤ 0 will be forced to 1.

R (RESET) =

Reset input, type BOOLEAN. This is an optional parameter. The default is FALSE. When this input is TRUE, the following occur: the internal storage areas are initialized, OUTPUT is set to zero, VALID_DATA is set FALSE, MAX_VALUE is set to -32768, and MIN_VALUE is set to 32767.

I (INPUT) =

Signal input, type INTEGER. This parameter must be specified as a numeric symbol only (literal value not accepted).

Outputs

DV (DATA_VALID) =

Data valid output, type BOOLEAN. This is an optional parameter. The DATA_VALID output is set TRUE only during the scan when the outputs are being updated (the RS scan). The DATA_VALID output is always set FALSE when RESET is TRUE.

MX (MAX_VALUE) =

Maximum value output, type INTEGER. This is an optional parameter. The parameter MAX_VALUE is updated once every RS scan with the maximum value of the samples read for INPUT during the last RS scan. When RESET is TRUE, MAX_VALUE is set to -32768.

MN (MIN_VALUE) =

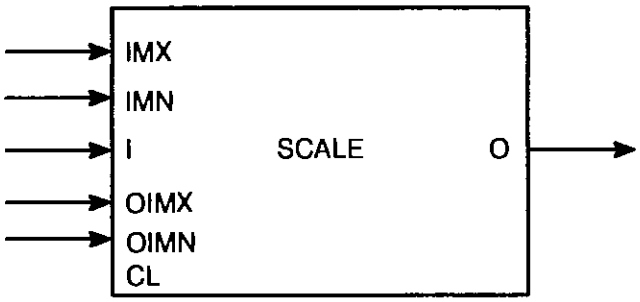
Minimum value output, type INTEGER. This is an optional parameter. The parameter MIN_VALUE is updated once every RS scan with the minimum value of the samples read for INPUT during the last RS scan. When RESET is TRUE, MIN_VALUE is set to 32767.

O (OUTPUT) =

Data output, type INTEGER. This parameter must be specified. OUTPUT is updated once every RS scans with the average of the samples read for INPUT during the last RS scan. When RESET is TRUE, OUTPUT is set to a value of zero.

28.0 SCALE

This function can be used in AutoMax Control Block tasks and UDC Control Block tasks.



Function

If CLAMP is TRUE and INPUT exceeds INPUT_MAX or INPUT_MIN, then the value of INPUT (not the actual input itself) is clamped at the proper limit.

$$\text{OUTPUT} = \frac{(\text{INPUT} - \text{INPUT\_MIN}) * (\text{OUTPUT\_IMAX} - \text{OUTPUT\_IMIN})}{\text{INPUT\_MAX} - \text{INPUT\_MIN}} + \text{OUTPUT\_IMIN}$$

Program Statement

```
CALL SCALE(INPUT=input%, CLAMP=clamp@,                    &
    INPUT_MAX=input_max%,                                 &
    INPUT_MIN=input_min%,                                 &
    OUTPUT_IMAX=output_imax%,                             &
    OUTPUT_IMIN=output_imin%,                             &
    OUTPUT=output%)
```

Inputs

CL (CLAMP) =

Clamp input, type BOOLEAN. This parameter is optional. The default is FALSE. If CLAMP is TRUE and INPUT exceeds either INPUT_MAX or INPUT_MIN, then clamp the value of INPUT at the proper limit. If CLAMP is FALSE, then the value of INPUT is not clamped.

I (INPUT) =

Signal input, type INTEGER. This parameter must be specified as a numeric symbol only (literal value not accepted).

IMX (INPUT_MAX) =

Maximum value for INPUT, type INTEGER. This parameter must be specified. If the value of INPUT is greater than INPUT_MAX and CLAMP is TRUE, then INPUT is clamped at INPUT_MAX.

IMN (INPUT_MIN) =

Minimum value for INPUT, type INTEGER. This parameter must be specified. If the value of INPUT is less than INPUT_MIN and CLAMP is TRUE, then INPUT is clamped at INPUT_MIN.

OIMX (OUTPUT_IMAX) =

Value for OUTPUT when INPUT is equal to INPUT_MAX, type INTEGER. This parameter must be specified.

OIMN (OUTPUT_IMIN) =

Value for OUTPUT when INPUT is equal to INPUT_MIN, type INTEGER. This parameter must be specified.

Outputs

O (OUTPUT) =

Data output, type INTEGER. This parameter must be specified. The value of OUTPUT is determined by the current values of INPUT, INPUT_MAX, INPUT_MIN, OUTPUT_IMAX, and OUTPUT_IMIN. See the functional description above.

28.1 Overflow Handling

If the computed answer for OUTPUT exceeds +32767 or -32768, then OUTPUT will be clamped to +32767 or -32768 and a run time error will be logged.

28.2 Application Notes

A typical use for the SCALE block would be to convert a 4 to 20 milliamp signal from a M/N 57C409 2 Channel Analog Input module to a linear signal with the range of 0 to 10000 counts to be used with the PID block.

The 4 to 20 mA input would normally require a 511 ohm load resister to convert the current signal into a voltage signal so that it can be read by the input module. At 4 mA the voltage signal would approximate 2.00 VDC and at 20 mA the voltage signal would approximate 10.00 VDC.

If the A/D input card produces a count value of 4095 for a 10.00 VDC input, then a 4 mA signal would be 819 counts and a 20 mA signal would be 4095 counts. See figure 28.1.

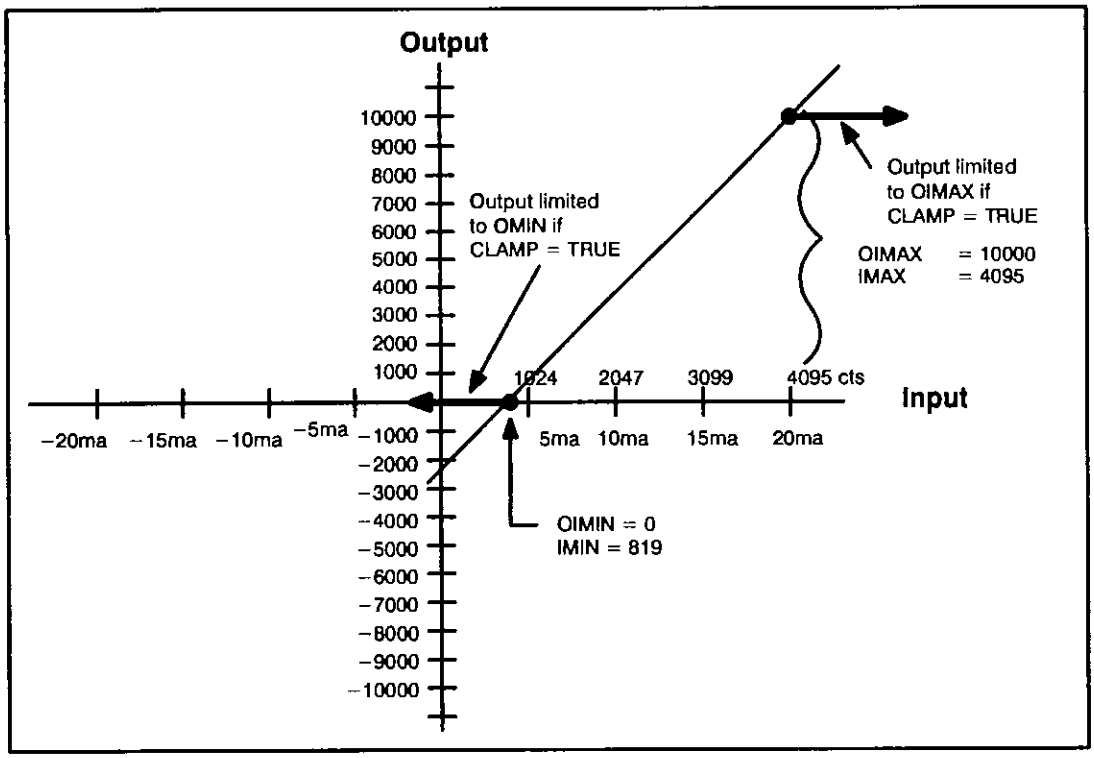


Figure 28.1 - Converting 4-20ma to 0-10000 counts

The SCALE block would be programmed as follows

```
nnnn CALL SCALE(INPUT = PROCESS_INPUT%,                    &
     INPUT_MAX = 4095,                                     &
     INPUT_MIN = 819,                                      &
     OUTPUT_IMAX = 10000,                                  &
     OUTPUT_IMIN = 0,                                      &
     OUTPUT = PROCESS_OUTPUT%)
```

If you want to limit the value of OUTPUT to the range specified by OUTPUT_IMAX and OUTPUT_IMIN even when the value of INPUT exceeds INPUT_MAX or INPUT_MIN, then the current value of the input CLAMP must be TRUE.

29.0 SEARCH

This function can be used in AutoMax Control Block tasks and UDC Control Block tasks.

Function

Compare INPUT against selected TABLE elements.

Search the selected table for a match according to the comparison options selected by the three BOOLEAN inputs COMPARE_GTR, COMPARE_EQU and COMPARE_LES. The search starts at the top of the table (array element 0) and tests INPUT against each element in the table until either a match is found or the end of the table is reached (last element in the array). If a match occurs, the search function is terminated and the index into the table where the match occurred is written to OUTPUT. FOUND is then set true. If no match occurred, then OUTPUT is set to a value of -1 and FOUND is set FALSE.

Program Statement

```
CALL SEARCH (COMPARE_GTR=<boolean literal>,                    &
    COMPARE_EQU=<boolean literal>,                             &
    COMPARE_LES=<boolean literal>,                             &
    INPUT=input%,                                              &
    SELECT=select@,                                            &
    TABLE1=table1%, TABLE2=table2%,                            &
    FOUND=found@, OUTPUT=output%)
```

Inputs

I (INPUT) =

Data input, type INTEGER. This parameter must be specified as a numeric symbol only (literal value not accepted).

GT (COMPARE_GTR) =

Compare INPUT > TABLE(n), type BOOLEAN. This is an optional parameter. The default for this parameter is FALSE. It must be entered explicitly as a boolean literal.

EQ (COMPARE_EQU) =

Compare INPUT = TABLE(n), type BOOLEAN. This is an optional parameter. The default for this input is FALSE. It must be entered explicitly as a boolean literal.

LT (COMPARE_LES) =

Compare INPUT < TABLE(n), type BOOLEAN. This is an optional parameter. The default for this parameter is FALSE. It must be entered explicitly as a boolean literal.

T1 (TABLE1) =

Search table #1, type INTEGER. This parameter must be specified as a variable name only (literal value not accepted). TABLE1 must be defined as a single dimension INTEGER array of length N. This array definition is used when you define the variable using a LOCAL or COMMON statement in the task.

SE (SELECT) =

This input selects the TABLE that is to be searched, type BOOLEAN. The default is FALSE. When FALSE, TABLE1 will be searched, else TABLE2 will be searched. If this parameter is specified, then parameter TABLE2 must also be specified.

T2 (TABLE2) =

Search table #2, type INTEGER. This parameter is optional. If specified it must be a variable name only (literal value is not accepted). If TABLE2 is specified then the parameter SELECT must also be specified. TABLE2 must be defined as a single dimension INTEGER array. The length of TABLE2 must be equal to TABLE1. This array definition is used when you define the variable using a LOCAL or COMMON statement in the task.

Outputs

O (OUTPUT) =

Data output, type INTEGER. This parameter must be specified. If a match occurs, OUTPUT is set equal to the array element (0 to N) in the table that met the search requirement. If no match occurs, then OUTPUT is set equal to -1.

F (FOUND) =

Found match output, type BOOLEAN. This is an optional parameter. This output is TRUE if an element in the selected table matched the INPUT according to the selected comparison parameters (COMPARE_GTR, COMPARE_EQU or COMPARE_LES). If no match occurs, this output is set FALSE.

Notes

1. The order in which the comparison options are programmed is unimportant. However, the number of comparison options

selected (programmed TRUE) must be a minimum of one and a maximum of two. If this requirement is not met, a compilation error will occur.

2. TABLE1 (and TABLE2 if specified) must be defined as a single dimension INTEGER array whose size (number of elements) is defined in a variable definition statement (LOCAL or COMMON) (e.g. LOCAL TABLE1%(16)). The size of TABLE2 if specified must be the same as TABLE1. The SEARCH block will begin searching through the selected table at the first element in the array, which is element 0, not element 1.

3. When loading data into the table(s) it may be necessary to manually sort the data in ascending or descending order.

   Example 1: If COMPARE_GTR mode has been specified, then sort the data in descending order.

INPUT = 15	TABLE 1%(0) = 40
	TABLE 1%(1) = 30
	TABLE 1%(2) = 20
then	TABLE 1%(3) = 10

   OUTPUT = 3 and FOUND = TRUE

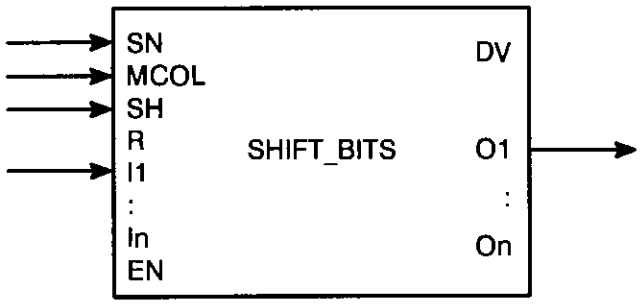
   Example 2: If COMPARE_LES mode has been specified, then sort the data in ascending order.

INPUT = 1501	TABLE 1%(0) = 100
	TABLE 1%(1) = 200
	TABLE 1%(2) = 300
then	TABLE 1%(3) = 400

   OUTPUT = 1 and FOUND = TRUE

30.0 SHIFT BITS

This function can be used in AutoMax Control block tasks only. It cannot be used in UDC Control Block tasks.



Maximum 'n' = 8

Function

When RESET is FALSE and SHIFT is TRUE, shift the data in the specified BOOLEAN data structure towards the output(s), update the output(s) with the state(s) at column MCOL - 1 and if ENABLE is TRUE shift the state(s) of the input(s) into column 0, else set the state(s) of column 0 FALSE.

Program Statement

```
CALL SHIFT_BITS(STRUCTURE_NAME=struc_name@,                    &
    MAX_COLUMNS=max_columns%,                                  &
    RESET=reset@, SHIFT=shift@,                                &
    INPUT1=input1@, ...INPUTn=inputn@,                         &
    ENABLE=enable@, DATA_VALID=data_valid@,                    &
    OUTPUT1=output1@, ... OUTPUTn=outputn@)
```

Inputs

SN (STRUCTURE_NAME) =

Name of the BOOLEAN data structure used to store the shifted data. This parameter must be specified by name only (literal value not accepted). The data structure name is limited to a maximum length of 15 characters and must be type BOOLEAN. The required BOOLEAN data structure is automatically created by this control block.

MCOL (MAX_COLUMNS) =

Required number of columns (depth) for the BOOLEAN data structure, type INTEGER. This parameter must be entered explicitly as a numeric literal. The minimum value is 1 and the maximum value is 32767 (see note 3). The columns of the data structure are numbered from 0 to MCOL - 1. Column 0 corresponds to the input of data structure and column MCOL - 1 to the output.

R (RESET) =

Reset input, type BOOLEAN. This parameter is optional. The default for this parameter is FALSE. When TRUE, all data in the data structure will be zeroed.

SH (SHIFT) =

Shift input, type BOOLEAN. This parameter must be specified. When RESET is FALSE and SHIFT is TRUE, shift the data in the data structure towards the output(s), update the output(s) with the state(s) at column MCOL - 1, and shift new states(s) into column 0 (see ENABLE).

EN (ENABLE) =

Enable input, type BOOLEAN. This parameter is optional. The default for this parameter is TRUE. When ENABLE and SHIFT are TRUE, the state(s) of the input(s) will be transferred into column 0 of the data structure. When ENABLE is FALSE and SHIFT is TRUE, set the state(s) of column 0 FALSE.

I1 (INPUT1) =

Data input 1, type BOOLEAN. This parameter must be specified.

In (INPUTn) =

Data input n, type BOOLEAN. A maximum of 8 inputs can be specified. Each specified input must have a corresponding output.

Outputs

DV (DATA_VALID) =

Data valid output, type BOOLEAN. This is an optional parameter. DATA_VALID is set TRUE when the data in the data structure has been shifted a minimum of MCOL times since RESET went FALSE. When RESET is TRUE, DATA_VALID is set FALSE.

O1 (OUTPUT1) =

Data output 1, type BOOLEAN. This parameter must be specified.

On (OUTPUTn) =

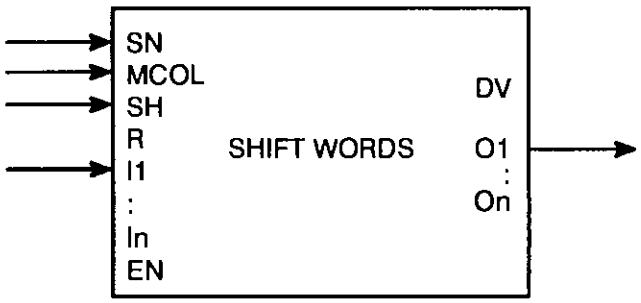
Data output n, type BOOLEAN. This is an optional parameter. The number of outputs specified must be equal to the number of inputs.

Notes

1. Data structure names are limited to a maximum length of 16 characters. If this requirement is not met, a compilation error will occur.
2. The order in which the input/output pairs are entered is unimportant. However, for every INPUT(n) programmed an OUTPUT(n) must also be programmed, and the input/output pairs must be contiguous beginning with input/output pair 1. If these requirements are not met, a compilation error will occur.
3. The SHIFT_BIT block creates a BOOLEAN data structure that is used to store the data shifted from the inputs towards the outputs. The size of the data structure in bytes is equal to MAX_COLUMNS. If the data structure size exceeds 32767 bytes, a compilation error will occur. Since the data structure is local to the control block task, its size is further limited by the maximum size of the total data storage area that the task can allocate for all the local variables.

31.0 SHIFT WORDS

This function can be used in AutoMax Control Block tasks only. It cannot be used in UDC Control Block tasks.



Maximum 'n' = 8

Function

When RESET is FALSE and SHIFT is TRUE, shift the data in the specified INTEGER data structure towards the output(s), update the output(s) with the value(s) at column MCOL - 1, and if ENABLE is TRUE, shift the data at the input(s) into column 0, else shift zeroes into column 0.

Program Statement

```
CALL SHIFT_WORDS(STRUCTURE_NAME=struc_name%,          &
    MAX_COLUMNS=max_columns%,                         &
    RESET=reset@, SHIFT=shift@,                       &
    INPUT1=input1%, ...INPUTn=inputn%,                &
    ENABLE=enable@, DATA_VALID=data_valid@,           &
    OUTPUT1=output1%, ...OUTPUTn=outputn%)
```

Inputs

SN (STRUCTURE_NAME) =

Name of the INTEGER data structure used to store the shifted data. This parameter must be specified by name only (literal value not accepted). The data structure name is limited to a maximum length of 16 characters and must be type INTEGER. The required INTEGER data structure is automatically created by this control block.

MCOL (MAX_COLUMNS) =

Required number of columns (depth) for the INTEGER data structure, type INTEGER. This parameter must be entered explicitly as a numeric literal. The minimum value is 1 and the maximum value is 32767 (see note 3). The columns of the data structure are numbered from 0 to MCOL - 1. Column 0 corresponds to the input of data structure and column MCOL - 1 to the output.

R (RESET) =

Reset input, type BOOLEAN. This parameter is optional. The default for this parameter is FALSE. When TRUE, all data in the data structure will be zeroed.

SH (SHIFT) =

Shift input, type BOOLEAN. This parameter must be specified. When RESET is FALSE and SHIFT is TRUE, shift the data in the data structure towards the output(s), update the output(s) with the value(s) at column MCOL - 1, and shift new data into column 0 (see ENABLE).

EN (ENABLE) =

Enable input, type BOOLEAN. This parameter is optional. The default for this parameter is TRUE. When ENABLE and SHIFT are TRUE, the values at the input(s) will be transferred into column 0 of the data structure. When ENABLE is FALSE and SHIFT is TRUE, zeros will be transferred into column 0.

I1 (INPUT1) =

Data input 1, type INTEGER. This parameter must be specified.

In (INPUTn) =

Data input n, type INTEGER. A maximum of 8 inputs can be specified. Each specified input must have a corresponding output.

Outputs

DV (DATA_VALID) =

Data valid output, type BOOLEAN. This is an optional parameter. DATA_VALID is set TRUE when the data in the data structure has been shifted a minimum of MCOL times since RESET went FALSE. When RESET is TRUE, DATA_VALID is set FALSE.

O1 (OUTPUT1) =

Data output 1, type INTEGER. This parameter must be specified.

On (OUTPUTn) =

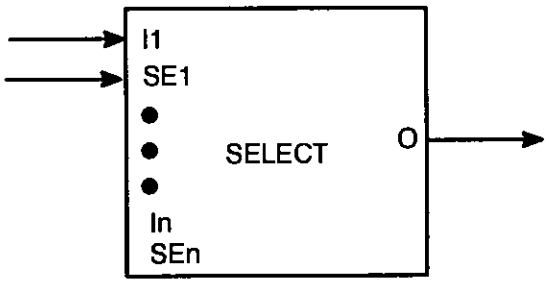
Data output n, type INTEGER. This is an optional parameter. The number of outputs specified must be equal to the number of inputs.

Notes

1. Data structure names are limited to a maximum length of 16 characters. If this requirement is not met, a compilation error will occur.
2. The order in which the input/output pairs are entered is unimportant. However, for every INPUT(n) programmed a OUTPUT(n) must also be programmed. In addition, the input/output pairs must be contiguous beginning with input/output pair 1. If these requirements are not met, a compilation error will occur.
3. The SHIFT_WORDS block creates an INTEGER data structure that is used to store the data shifted from the inputs towards the outputs. The size of the data structure in bytes is equal to two times the number of programmed inputs times the number of MAX_COLUMNS specified. If the data structure size exceeds 32767 bytes, a compilation error will occur. Since the data structure is local to the control block task, its size is further limited by the maximum size of the total data storage area that the task can allocate for all the local variables.

32.0 SELECT

This function can be used in AutoMax Control Block tasks and UDC Control Block tasks.



Maximum 'n' = 8

Function

OUTPUT will equal the sum of all selected inputs. If no SELECT is TRUE, OUTPUT = 0.

Program Statement

```
CALL SELECT(INPUT1 = input1%,                    &
     SELECT1 = select1@....,                     &
     INPUTn = inputn%,                           &
     SELECTn = selectn@,                         &
     OUTPUT = output%)                           &
```

Inputs

I1 (INPUT1) =

INTEGER signal input 1. This parameter must be specified.

SE1 (SELECT1) =

BOOLEAN signal 1 select. This parameter must be specified. When TRUE, INPUT1 is algebraically summed with other selected inputs, producing OUTPUT.

In (INPUTn) =

INTEGER signal input n. This is an optional parameter. A maximum of 8 inputs can be specified.

SEn (SELECTn) =

BOOLEAN signal select. This is an optional parameter. However, a SELECT is required for every INPUT. When TRUE, (INPUTn) is algebraically summed with other selected inputs, producing OUTPUT.

Outputs

O (OUTPUT) =

INTEGER signal output. This parameter must be specified.

Notes

1. The order in which (INPUTn) and (SELECTn) pairs are entered is not important. However, it is required that, for "m" channels used, channels 1 through "m" be programmed. In other words, the channels used must be contiguous beginning with channel 1. A compilation error will occur if this requirement is not met.

   The following is a correct statement:

```
CALL SELECT (INPUT1 = INA%,SELECT1 = SELECTA@,      &
    INPUT2 = INB%,SELECT2 = SELECTB@,               &
    INPUT3 = INC%,SELECT3 = SELECTC@,               &
    INPUT4 = IND%,SELECT4 = SELECTD@,               &
    OUTPUT = output%)
```

   The following is also correct, illustrating that the order of entry is not important:

```
CALL SELECT (INPUT2 =  INB%,SELECT2 = SELECTB@,     &
    INPUT4 = IND%,SELECT4 = SELECTD@,               &
    INPUT3 = INC%,SELECT3 = SELECTC@,               &
    INPUT1 = INA%,SELECT1 =  SELECTA@,              &
    OUTPUT= output%)
```

   The following is an incorrect statement because the channels are not contiguous:

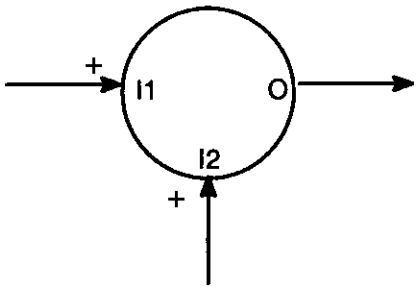
```
CALL SELECT (INPUT1 = INA%,SELECT1 = SELECTA@,      &
    INPUT3 = INC%,SELECT3=SELECTC@,                 &
    INPUT4 = IND%,SELECT4=SELECTD@,                 &
    OUTPUT = output%)
```

2. Overflow Handling: During block execution, (INPUTn) and (SELECTn) pairs are read and processed in sequential order beginning with INPUT1 and SELECT1. Since the sum of the selected inputs is calculated as a 32-bit value and inputs are integer quantities (16-bit values), an overflow cannot occur during these intermediate additions (given an 8 input channel restriction).

   When all consecutive sequential input channels are processed, the final sum (OUTPUT) must be within the range −32768 to +32767 or an error will be logged and the result will be clamped to −32768 or +32767, producing OUTPUT.

33.0 SUMMER

This function can be used in AutoMax Control Block tasks and UDC Control Block tasks.



Function

OUTPUT = INPUT1 + INPUT2

Program Statement

```
CALL SUMMER(INPUT1 = input1%,                    &
    INPUT2 = input2%,                            &
    OUTPUT = output%)
```

Inputs

I1 (INPUT1) =

INTEGER signal input 1. This parameter must be specified.

I2 (INPUT1) =

INTEGER signal input 2. This parameter must be specified.

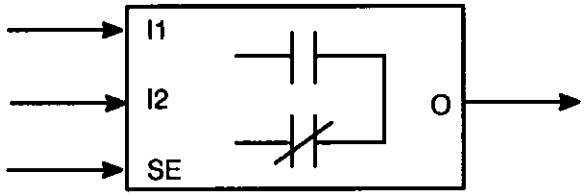
Outputs

O (OUTPUT) =

INTEGER signal output. This parameter must be specified. OUTPUT is limited to the range −32768 to +32767. If this limit is exceeded, an error will be logged.

34.0 SWITCH

This function can be used in AutoMax Control Block tasks and UDC Control Block tasks.



Function

If SELECT = TRUE then OUTPUT = INPUT1
else

OUTPUT = INPUT2

Program Statement

```
CALL SWITCH(INPUT1 = input1%,                    &
    INPUT2 = input2%,                            &
    SELECT = select@,                            &
    OUTPUT=output%)
```

Inputs

I (INPUT1) =

INTEGER signal input 1. This parameter must be specified.

I2 (INPUT2) =

INTEGER signal input 2. This parameter must be specified.

SE (SELECT) =

BOOLEAN signal select. This parameter must be specified. When TRUE, INPUT1 is connected to OUTPUT. When FALSE, INPUT2 is connected to OUTPUT.

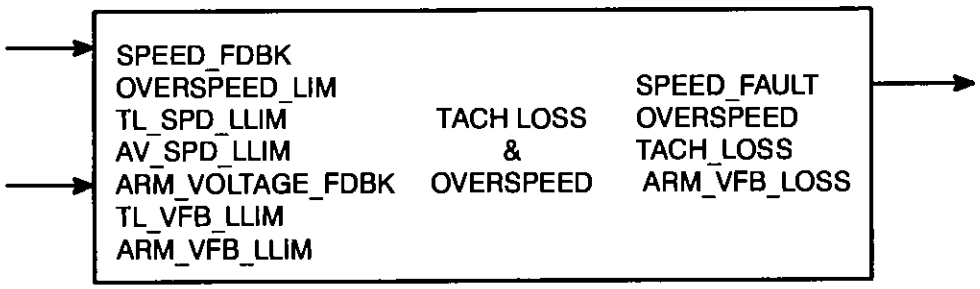
Outputs

O (OUTPUT) =

INTEGER signal output. This parameter must be specified.

35.0 TACH LOSS AND OVERSPEED

This function can be used in UDC Control Block tasks only. It cannot be used in AutoMax Control Block tasks.



Function

This function provides tachometer loss and overspeed detection using speed feedback and armature voltage for Distributed Power System drives. Use of this block is appropriate for control of motor-generator sets or in speed regulators where a signal proportional to speed that is independent of the device measuring speed (such as a resolver) is not provided by the PMI.

Program Statement

```
CALL TACHLOSS_OVERSPEED( SPEED_FDBK = speed_fdbk%,   &
     ARM_VOLTAGE_FDBK = arm_voltage_fdbk%,           &
     OVERSPEED_LIM = overspeed_lim%,                 &
     TL_SPD_LLIM = tl_spd_llim%,                     &
     AV_SPD_LLIM = av_spd_llim%,                     &
     TL_VFB_LLIM = tl_vfb_llim%,                     &
     ARM_VFB_LLIM = arm_vfb_llim%,                   &
     SPEED_FAULT = speed_fault@,                     &
     OVERSPEED = overspeed@,                         &
     TACH_LOSS = tach_loss@,                         &
     ARM_VFB_LOSS = arm_vfb_loss@ )
```

Inputs

SPEED_FDBK =

INTEGER speed feedback from the drive (typically 4095 = gear-in speed). This parameter must be specified. There is no default.

OVERSPEED_LIM =

INTEGER overspeed trip point or limit (typically 10% over gear-in speed). The default is 4505.

TL_SPD_LLIM =

INTEGER speed feedback low limit used for tach loss detection (typically 5% of gear-in speed). The default is 205.

AV_SPD_LLIM =

INTEGER speed feedback low limit used for armature voltage feedback loss detection (typically 40% of gear-in speed). The default is 1638.

ARM_VOLTAGE_FDBK =

INTEGER armature voltage feedback from drive (typically 3000 = rated armature volts). This parameter must be specified. There is no default.

TL_VFB_LLIM =

INTEGER armature voltage feedback low limit used for tach loss detection (typically 40% of rated voltage). The default is 1200.

ARM_VFB_LLIM =

INTEGER armature voltage feedback low limit used for armature voltage feedback loss detection (typically 5% of rated voltage). The default is 150.

Outputs

SPEED_FAULT =

BOOLEAN speed emergency stop output; TRUE when overspeed, tach loss, or armature voltage feedback loss is detected. This parameter must be specified.

OVERSPEED =

BOOLEAN output TRUE when overspeed is detected. The default is FALSE.

TACH_LOSS =

BOOLEAN output TRUE when tach loss is detected. The default is FALSE.

ARM_VFB_LOSS =

BOOLEAN output TRUE when armature voltage feedback loss is detected. The default is FALSE.

Notes

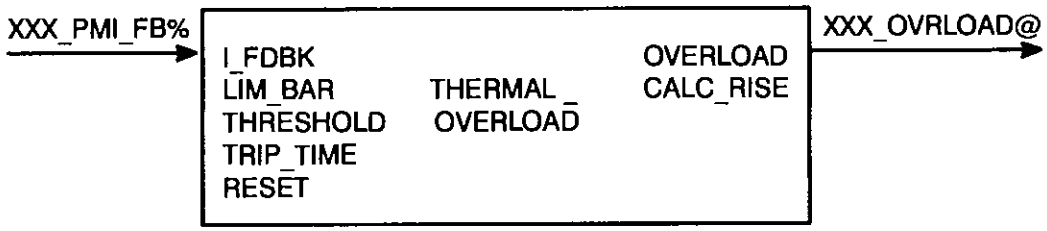
1. If armature voltage feedback comes from the PMI in volts, the armature volts low limit must also be in volts.

35.1 Setup Calculations and Block Equations

```
IF ABS( XXX_SPD_FBK% ) > OVERSPD_LIM% THEN
    XXX_OVERSPD@ = TRUE
ELSE
    XXX_OVERSPD@ = FALSE
END_IF

IF ABS( XXX_SPD_FBK% ) < TL_SPD_FBK_LLIM% AND
    ABS( XXX_ARM_VFB% ) > TL_VFB_LOW_LIM% THEN
        XXX_TACH_LOSS@ = TRUE
ELSE
    XXX_TACH_LOSS@ = FALSE
END_IF

IF ABS( XXX_SPD_FBK% ) > AV_SPD_FBK_LLIM% AND
    ABS( XXX_ARM_VFB% ) < ARM_VFB_LOW_LIM% THEN
        XXX_AVFB_LOSS@ = TRUE
ELSE
    XXX_AVFB_LOSS@ = FALSE
END_IF

IF XXX_OVERSPD@   = TRUE OR &
    XXX_TACH_LOSS@ = TRUE OR &
    XXX_AVFB_LOSS@ = TRUE THEN
        XXX_SPD_ESTOP@ = TRUE
ELSE
    XXX_SPD_ESTOP@ = FALSE
END_IF
```

36.0 THERMAL OVERLOAD

This function can be used in UDC Control Block tasks only. It cannot be used in AutoMax Control Block tasks.



Function

This block is used to create a model of the temperature in a device such as a single motor or Power Module controlled by a Distributed Power System drive and to turn on an alarm when an overload condition exists. The block calculates a rise in temper ature based on current feedback. When operating above 100%, if the rise in temperature exceeds the programmed limit, the OVERLOAD output will turn on. After the overload condition is detected, the rise in temperature must return to the 100% condition before the drive will be allowed to turn on again.

Note that this block is required in all UDC Control Block tasks, unless motor thermal overload protection is provided by a hardware device.

Program Statement

```
CALL THERMAL_OVERLOAD( I_FDBK = xxx_pmi_fb%,          &
    LIM_BAR = lim_bar%,                               &
    THRESHOLD = threshold%,                           &
    TRIP_TIME = trip_time%,                           &
    RESET = reset@,                                   &
    OVERLOAD = xxx_ovrload@,                          &
    CALC_RISE = calc_rise% )
```

Inputs

I_FDBK =

This variable is the current feedback as sampled by the PMI (Power Module Interface), register 211/1211 in the UDC dual port for AC drives, and 207/1207 for DC drives. It is scaled so 4095 counts is the maximum current that will be produced. This variable must be defined as a common integer. Recall that there must be no duplicate common variable names in an AutoMax rack; each instance of this variable name in a rack must be customized. The suggested name of the variable is XXX_PMI_FB%, with XXX used for UDC slot and drive designation, for example: S12A_PMI_FB% might be used for the current feedback value returned by the PMI connected to port A of the UDC in slot 12.

LIM_BAR=

This variable is the maximum current limit for the motor in percent. The value used for LIM_BAR must be the same value entered for the Motor Overload Ratio during drive parameter configuration. It defines what percent of motor current is expressed by 4095 counts. For example, if LIM_BAR is 150, then 100% motor current is 2730 counts. If LIM_BAR is 200, then 100% motor current is 2047 counts. This variable may be programmed as an integer variable, an integer constant, or it may be omitted, in which case it defaults to a value of 150. Legal values range from a low of 115 to a high of 400. If the value is outside this range, it will be limited to this range and an error will be logged. The value must be greater than the value of THRESHOLD or an error will be logged.

THRESHOLD =

This variable determines the threshold of current at which the overload output is turned on. For example, if threshold is set to 114, then an overload will be detected when steady state current reaches 115%. This variable may be programmed as an integer variable or an integer constant, or it may be omitted, in which case it defaults to a value of 114. Legal values range from a low of 110 to a high of 124. If the value is outside this range, it will be limited to this range and an error will be logged. The value must be less than the value of LIM_BAR or an error will be logged.

TRIP_TIME =

This variable is the trip time in seconds. If I_FDBK is at steady state 100%, and then stepped to LIM_BAR, the OVERLOAD output will turn on in TRIP_TIME seconds. This variable may be programmed as an integer variable or an integer constant, or it may be omitted, in which case it defaults to a value of 60. Legal values range from a low of 10 to a high of 120. If the value is outside this range, it will be limited to this range and an error will be logged.

RESET=

This variable is the reset input. When this input is turned on, the OVERLOAD output is turned off, and CALC_RISE is set to zero. This parameter is intended only for test purposes and should not be used in application tasks controlling real applications. If this variable is programmed, it must be specified as a boolean.

Outputs

OVERLOAD =

This variable is a boolean that will be turned on when an overload is detected. This boolean must be programmed in a Ladder Logic task to turn off the drive when the bit turns on. It must be defined as a common boolean. Recall that there must be no duplicate common variable names in an AutoMax rack; each instance of this variable name in a rack must be customized. The suggested name of the variable is XXX_OVRLOAD@, with XXX used for UDC slot and drive designation, for example: S07B_PMI_FB% might be used for the bit that will turn off the drive connected to port B on the UDC module in slot 7.

CALC_RISE =

This variable displays the calculated rise in temperature. The calculation squares I_FDBK, scales it, and then applies a low pass filter to introduce a time delay. For example, when I_FDBK is at steady state 100%, if I_FDBK is then stepped to 110%, CALC_RISE will reach a steady state value of 1210, (N%**2/10). With THRESHOLD at 114%, the trip point will be 1300. If I_FDBK remains less than 114%, CALC_RISE will remain less than 1300 and OVERLOAD will not turn on.

If this variable is programmed, it must be defined as an integer.

36.1 Setup Calculations and Block Equations

The following setup calculations are performed internally by the Thermal Overload function based on the parameters entered.

$$Wlg = \frac{\ln[\ (LIM\_BAR^{**}2 - 100^{**}2) / (LIM\_BAR^{**}2 - THRESHOLD^{**}2)\ ]}{TRIP\_TIME\ (sec.)}$$

```
SCALE = (4095 / LIM_BAR)**2 * 10

IF OVERLOAD = TRUE THEN
    LIMIT = 1000
ELSE
    LIMIT = THRESHOLD**2 / 10
END_IF
```

$$CALC\_RISE = \frac{(I\_FDBK^{**}2)}{SCALE} * \frac{1}{(1 + s / \omega lg)}$$

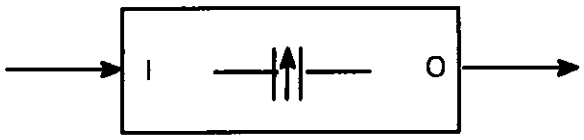
```
IF RISE% > LIMIT THEN
    OVERLOAD = TRUE
ELSE
    OVERLOAD = FALSE
END_IF
```

36.2 Special Notes

1. UL 508C section 56.1.3 specifies that when subjected to 200% of rated full load motor current, the overload protection must trip in at least eight (8) minutes. Because TRIP_TIME is calibrated from 100% to current limit, and TRIP_TIME from zero to current limit is approximately four times longer, the maximum trip time that is allowed is 2 minutes (120 seconds). To meet UL listing requirements, any value for TRIP_TIME greater than 120 seconds is limited to 120 seconds.
2. The National Electric Code (430-32; 1993) requires that thermal overloads protecting motors having a 1.0 service factor trip at load currents no greater than 115% of full load. To meet NEC, the THRESHOLD block parameter has a default value of 114%. If your motor has a service factor greater than 1.0, you may use a value up to 124.

37.0 TRANSITION

This function can be used in AutoMax Control Block tasks and UDC Control Block tasks.



Function

OUTPUT = TRUE when INPUT goes from off to on

Program Statement

CALL TRANSITION(INPUT = input@, OUTPUT = output@)

Inputs

I (INPUT) =

BOOLEAN signal input. This parameter must be specified. It must be specified as a variable name only (literal value not accepted).

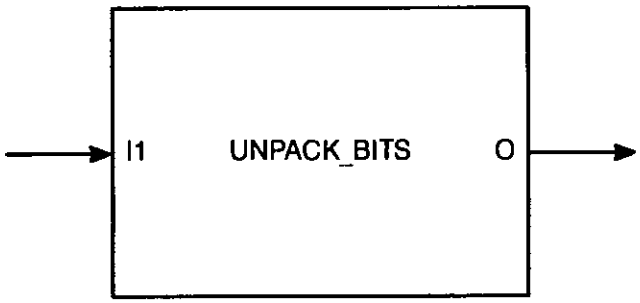
Outputs

O (OUTPUT) =

BOOLEAN signal output. This parameter must be specified.

38.0 UNPACK BITS

This function can be used in AutoMax Control Block tasks and UDC Control Block tasks.



'n' = 0...15

Function

OUTPUTn is set to the state of BITn in INPUT

Program Statement

```
CALL UNPACK_BITS(INPUT=input%,                                    &
    OUTPUT0=output0@, ... OUTPUTn=outputn@)
```

Inputs

I (INPUT) =

Signal input, type INTEGER. This parameter must be specified.

Outputs

On (OUTPUTn) =

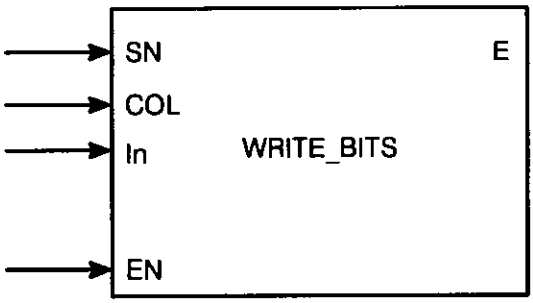
Data output 0...15, type BOOLEAN. The outputs can be specified in any order.

Notes

1. The order in which the outputs (output0...output15) are programmed is unimportant. However, a minimum of one output must be programmed. If this requirement is not met, a compilation error will occur.

39.0 WRITE BITS

This function can be used in AutoMax Control Block tasks only. It cannot be used in UDC Control Block tasks.



Maximum 'n' = 8

Function

This function stores data into a column in the specified BOOLEAN data structure.

Program Statement

```
CALL WRITE_BITS(STRUCTURE_NAME=structure_name@,        &
    COLUMN=column%, ENABLE=enable@,                    &
    INPUT1=input1@, ...INPUTN=inputn@,                 &
    ERROR=error@)
```

Inputs

SN (STRUCTURE_NAME) =

Name of the BOOLEAN data structure where data is to be written to. This parameter must be specified by name only (literal value not accepted). The data structure name is limited to a maximum length of 15 characters and must be type BOOLEAN. The specified data structure must be created by a control block within the task. Refer to the SHIFT_BITS block for an example of a control block that creates a BOOLEAN data structure.

COL (COLUMN) =

Selects a column within the specified BOOLEAN data structure, type INTEGER. This parameter is required. The columns are numbered from 0 to MCOL - 1, where MCOL is equal to the number of columns (depth) defined by the control block that created the data structure.

EN (ENABLE) =

Enable input, type BOOLEAN. This parameter is required. The state(s) of the input(s) are written into the column specified by COLUMN when ENABLE is TRUE. If this parameter is FALSE, no data will be written into the column.

In (INPUTn) =

Data input n, type BOOLEAN. The inputs can be specified in any order.

Outputs

E (ERROR) =

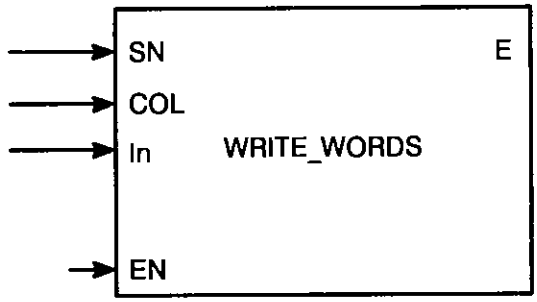
Error output, type BOOLEAN. This is an optional parameter. The output is TRUE if the value of COLUMN selects a non-existent column for the specified data structure, or if the column specified has not been loaded previously with data by the control block that created it. Valid values for COLUMN range from 0 to MCOL - 1. See COLUMN above.

Notes

1. The WRITE_BITS block must reference a BOOLEAN data structure that was created by a control block within the task. A minimum of one input must be programmed. The order in which the inputs (input1...input8) are programmed is unimportant. However, all of the inputs programmed by the WRITE_BITS block must also be defined by the control block that created the data structure. If these requirements are not met, a compilation error will occur.
2. If the value of COLUMN selects a non-existent column, the output ERROR is set TRUE, no data is stored in the specified data structure, and the appropriate run time error is logged.
3. If the value of COLUMN selects a column that has not been previously loaded with data by the control block that created it, the output ERROR is set TRUE, no data is stored in the specified data structure, but no run time error is logged.

40.0 WRITE WORDS

This function can be used in AutoMax Control Block tasks only. It cannot be used in UDC Control Block tasks.



Maximum 'n' = 8

Function

This function stores data into a column in the specified INTEGER data structure.

Program Statement

```
CALL WRITE_WORDS (STRUCTURE_NAME=struc_name%,        &
    COLUMN=column%, ENABLE=enable@,                  &
    INPUT1=input1%, ...INPUTN=inputn%,               &
    ERROR=error@)
```

Inputs

SN (STRUCTURE_NAME) =

Name of the INTEGER data structure where data is to be written to. This parameter must be specified by name only (literal value not accepted). The data structure name is limited to a maximum length of 15 characters and must be type INTEGER. The specified data structure must be created by a control block within the task. Refer to the SHIFT_WORDS block for an example of a control block that creates an INTEGER data structure.

COL (COLUMN) =

Selects a column within the specified INTEGER data structure, type INTEGER. This parameter is required. The columns are numbered from 0 to MCOL - 1, where MCOL is equal to the number of columns (depth) defined by the control block that created the data structure.

EN (ENABLE =

Enable input, type BOOLEAN. This parameter is required. The values read from the inputs are written into the column specified by COLUMN when ENABLE is TRUE. If this parameter is FALSE, no data will be written into the column.

In (INPUTn) =

Data input n, type INTEGER. The inputs can be specified in any order.

Outputs

E (ERROR) =

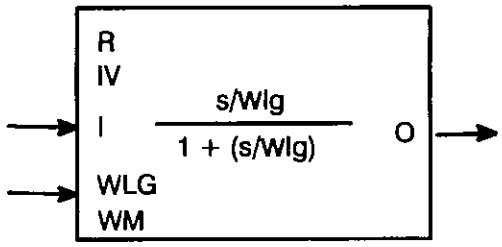
Error output, type BOOLEAN. This is an optional parameter. The output is TRUE if the value of COLUMN selects a non-existent column for the specified data structure or if the column specified has not been previously loaded with data by the control block that created it. Valid values for COLUMN range from 0 to MCOL - 1 (see COLUMN above).

Notes

1. The WRITE_WORDS block must reference an INTEGER data structure that was created by a control block within this task. A minimum of one input must be programmed. The order in which the inputs (input1 ...input8) are programmed is unimportant. However, all of the inputs programmed by the WRITE_WORDS block must also be defined by the control block that created the data structure. If these requirements are not met, a compilation error will occur.
2. If the value of COLUMN selects a non-existent column, the output ERROR is set TRUE, no data is stored in the specified data structure, and the appropriate run time error is logged.
3. If the value of COLUMN selects a column that has not been previously loaded with data by the control block that created it, the output ERROR is set TRUE, no data is stored in the specified data structure, but no run time error is logged.

41.0 DIFFERENTIATOR LAG

This function can be used in AutoMax Control Block tasks and UDC Control Block tasks.



Function

LAPLACE TRANSFER FUNCTION = $\frac{s/\omega lg}{1 + (s/\omega lg)}$

Program Statement

```
CALL DIFF_LAG(INPUT=input%,                    &
    WLG=wlg,                                   &
    WM=nnn.n,                                  &
    INITIAL_VALUE=initial_value%,              &
    RESET=reset@ ,                             &
    OUTPUT=output%)
```

Inputs

R (RESET) =

BOOLEAN device reset. The default for this parameter is FALSE. When this parameter is TRUE, OUTPUT will be held at INITIAL_VALUE.

IV (INITIAL_VALUE) =

INTEGER initial value. The default for this parameter is zero.

I (INPUT) =

INTEGER signal input. This parameter must be specified as a variable name only (literal value not accepted).

WLG (ωlg) =

REAL lag frequency in radians/second. This parameter must be specified. You must include a decimal point in the actual value.

WM (ωm) =

REAL mapping frequency in radians/second. If specified, this parameter must be entered explicitly as a real literal. The default value for this parameter is ωs divided by 20 where ωs is the frequency in rad/sec. You must include a decimal point in the actual value.

Outputs

O (OUTPUT) =

INTEGER signal output. This parameter must be specified.

41.1 DIFF_LAG ωm Limitations

ωm is equal to or greater than 0.0013872 divided by TICKS
ωm is equal to or less than 253.866 divided by TICKS.

$$\text{Low Limit} = \left(\frac{2}{T}\right) \text{ ARCTAN } (2^{-18})$$

$$= \left(\frac{2\,(2^{-18})}{T}\right)$$

$$= \frac{2^{-17}}{T}$$

$$= \frac{0.0013872}{\text{TICKS}}$$

$$\text{High Limit} = \frac{2\pi}{4.5T}$$

$$= \frac{253.866}{\text{TICKS}}$$

where:

T = scan period in seconds
= number of CPU clock ticks times tick rate
2^{-18} = 3.814697266E-06

Refer to section 51.0 Special Coefficient Restrictions, for further restriction notes.

41.2 DIFF_LAG ωlg Limitations

ωlg is equal to or greater than 0.00137573 divided by TICKS. ωlg is equal to or less than 256.055 divided by TICKS.

Low Limit $= C(2^{-18})$

$$= \frac{\omega m(2^{-18})}{\text{TAN}\left(\frac{\omega m * T}{2}\right)}$$

High Limit $= 0.71C$

$$= \frac{0.71\omega m}{\text{TAN}\left(\frac{\omega m * T}{2}\right)}$$

If ωm is defaulted (ωm = ωs ÷ 20):

$$\text{Low Limit} = (2^{-18}) \quad \frac{1.9835}{T}$$

$$= \frac{0.00137573}{\text{TICKS}}$$

$$\text{High Limit} = .071 \left(\frac{1.9835}{T}\right)$$

$$= \frac{256.055}{\text{TICKS}}$$

where:

T = scan period in seconds
= number of CPU clock ticks times tick rate

ωs = scan frequency in radians/second

$$= \frac{2\pi}{T}$$

C = bilinear mapping constant

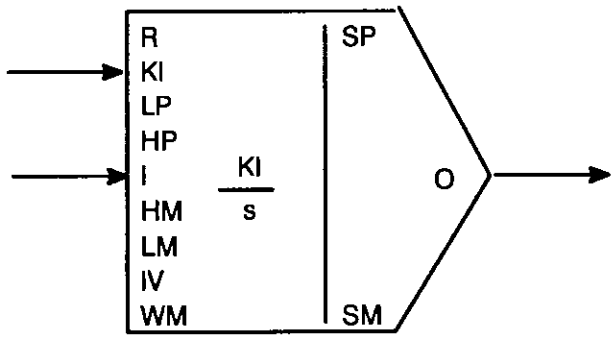
$$= \frac{\omega m}{\text{TAN}\left(\frac{\omega m * T}{2}\right)}$$

$2^{-18} = 3.814697266E\text{-}06$

Refer to section 51.0, Special Coefficient Restrictions, for further restriction notes.

42.0 INTEGRATE

This function can be used in AutoMax Control Block tasks and UDC Control Block tasks.



Function

LAPLACE TRANSFER FUNCTION = $\frac{KI}{s}$

Program Statement

```
CALL INTEGRATE(INPUT = input%,                          &
    KI = ki,                                            &
    WM = nnn.n,                                         &
    INITIAL_VALUE = initial_value%,                     &
    LIMIT_PLUS = limit_plus%,                           &
    LIMIT_MINUS = limit_minus%,                         &
    RESET = reset@,                                     &
    HOLD_PLUS = hold_plus@,                             &
    HOLD_MINUS = hold_minus@,                           &
    SATURATED_PLUS = saturated_plus@,                   &
    SATURATED_MINUS = saturated_minus@,                 &
    OUTPUT = output%)
```

Inputs

R (RESET) =

BOOLEAN integrator reset. The default for this parameter is FALSE. When this parameter is TRUE, OUTPUT will be held at INITIAL_VALUE.

KI (KI) =

REAL integrator gain. This parameter must be specified. You must include a decimal point in the actual value.

LP (LIMIT_PLUS) =

INTEGER integrator upper limit. The default for this parameter is 32767. This parameter will limit OUTPUT from becoming more positive. It will not prevent the output value from becoming more negative.

HP (HOLD_PLUS) =

BOOLEAN integrator hold plus. The default for this parameter is FALSE. When this parameter is TRUE,OUTPUT will be prevented from becoming more positive. It will be permitted to go more negative.

I (INPUT) =

INTEGER signal input. This parameter must be specified.

HM (HOLD_MINUS) =

BOOLEAN integrator hold minus. The default for this parameter is FALSE. When this parameter is TRUE, OUTPUT will be prevented from becoming more negative. It will be permitted to go more positive.

LM (LIMIT_MINUS) =

INTEGER integrator lower limit. The default for this parameter is −32767. This parameter will limit OUTPUT from becoming more negative. It will not prevent the output value from becoming more positive.

IV (INITIAL_VALUE) =

INTEGER initial value of integrator. The default for this parameter is zero.

WM (ωm) =

Mapping frequency in radians/second. If specified, this parameter must be entered explicitly as a REAL literal. The default value for this parameter is ωs divided by 20 where ωs is the frequency in rad/sec. You must include a decimal point in the actual value.

Outputs

SP (SATURATED_PLUS) =

BOOLEAN SATURATED plus output.This parameter is optional. TRUE if OUTPUT% reaches LIMIT(+).

O (OUTPUT) =

INTEGER signal output. This parameter must be specified.

SM (SATURATED_MINUS) =

BOOLEAN saturated minus output. This parameter is optional. TRUE if OUTPUT% reaches LIMIT(−).

42.1 INTEGRATE ωm Limitations

ωm is equal to or greater than 0.01. ωm is equal to or less than 0.9π divided by T.

Low Limit = .01

High Limit = $\frac{0.9\pi}{T}$

where:
T = scan period in seconds
= number of CPU clock ticks times tick rate

42.2 INTEGRATE K1 Limitations

K1 is equal to or greater than the value 0.00137573 divided by TICKS. K1 is equal to or less than 0.9π divided by T.

Low Limit = $C(2^{-18})$

$$= \frac{\omega m(2^{-18})}{\text{TAN}\left(\frac{\omega m * T}{2}\right)}$$

High Limit = $\frac{0.9\pi}{T}$

If ωm is defaulted ($\omega m = \omega s \div 20$):

Low Limit = $(2^{-18})\ \frac{1.9835}{T}$

where:
T = scan period in seconds
= number of CPU clock ticks times tick rate
ωs = scan frequency in radians/second
= $\frac{2\pi}{T}$
C = bilinear mapping constant
= $\frac{\omega m}{\text{TAN}\left(\frac{\omega m * T}{2}\right)}$
2^{-18} = 3.814697266E-06

Refer to section 51.0, Special Coefficient Restrictions, for further restrictions.

42.3 Calculating K1 for INTEGRATE Time Domain Applications

The following describes calculating K1 when the INTEGRATE block is used for its time domain characteristics. When used in this manner, the frequency characteristics of the block are of no concern. Therefore, the ωm specification will always be defaulted (not specified).

Calculate K1 such that OUTPUT will accumulate (change) by "y" counts/second with a constant "x" INPUT value and a scan period of "t" TICKS. The equation executed by the INTEGRATE block is as follows:

OUTPUT = Kx[INPUT + INPUT(n-1)] + OUTPUT(n-1)

where:

$$Kx = \frac{K1}{C}$$

$$c = \frac{\omega m}{\text{TAN}\left(\frac{\omega m * T}{2}\right)}$$

Since ωm is defaulted (ωm = ωs ÷ 20 where ωs = 2π ÷ T):

$$\omega m = \frac{2\pi}{20T} = \frac{\pi}{10T}$$

Therefore,

$$C = \frac{\left(\frac{\pi}{10T}\right)}{\left(\frac{\pi}{10T}\right)\left(\frac{T}{2}\right)}$$

$$= \frac{\left(\frac{\pi}{10T}\right)}{0.15838}$$

$$= \frac{1.9835}{T}$$

Determine the Counts/Scan as follows:

$$\text{Counts/Scan} = \frac{y\ \text{Counts/Second}}{\text{Scan/Second}}$$

$$= \left(\frac{y}{\frac{1}{T}}\right)$$

$$= y * T$$

Because OUTPUT = Counts/Scan = Kx(2x), solve for Kx:

$$Kx = \frac{(\text{Counts/Scan})}{2x}$$

$$= \frac{y * T}{2x}$$

To solve K1 in the formula $Kx = \frac{K1}{C}$:

$$K1 = Kx * C$$

$$= \left(\frac{y * T}{2x}\right)\left(\frac{1.9835}{T}\right)$$

$$= \left(\frac{y * 1.9835}{2x}\right)$$

$$= 0.99179\left(y/x\right)$$

For example, if an application required that the output integrate 1200 counts/second with a constant input value of 2000, we calculate for K1 as follows:

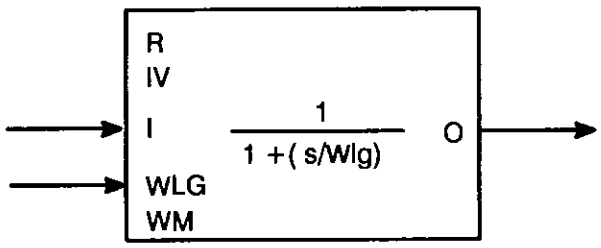
$$K1 = 0.99179 \left(\frac{y}{x}\right)$$

$$= 0.99179 \left(\frac{1200}{2000}\right)$$

$$= .59507$$

43.0 LAG

This function can be used in AutoMax Control Block tasks and UDC Control Block tasks.



Function

LAPLACE TRANSFER FUNCTION = $\frac{1}{1 + (s/\omega lg)}$

Program Statement

```
CALL LAG(INPUT = input%,                                    &
    WLG = wlg,                                              &
    WM = nnn.n,                                             &
    INITIAL_VALUE = initial_value%,                         &
    RESET = reset@,                                         &
    OUTPUT = output%)
```

Inputs

R (RESET) =

BOOLEAN device reset. The default for this parameter is FALSE. When this parameter is TRUE, OUTPUT will be held at INITIAL_VALUE.

IV (INITIAL_VALUE) =

INTEGER initial value. The default for this parameter is zero.

I (INPUT) =

INTEGER signal input. This parameter must be specified. It must be specified as a variable name only (literal value not accepted).

WLG (ωlg) =

REAL lag frequency in radians/second. This parameter must be specified. You must include a decimal point in the actual value.

WM (ωm) =

REAL mapping frequency in radians/second. If specified, this parameter must be entered explicity as a real literal. The default value for this parameter is ωs divided by 20 where ωs is the frequency in rad/sec. You must include a decimal point in the actual value.

Outputs

O (OUTPUT) =

INTEGER signal output. This parameter must be specified.

43.1 LAG ωm Limitations

ωm is equal to or greater than 0.0013872 divided by TICKS.
ωm is equal to or less than 253.866 divided by TICKS.

$$\text{Low Limit} = \left(\frac{2}{T}\right) \text{ARCTAN } (2^{-18})$$

$$= \left(\frac{2\ (2^{-18})}{T}\right)$$

$$= \frac{2^{-17}}{T}$$

$$= \frac{0.0013872}{\text{TICKS}}$$

$$\text{High Limit} = \frac{2\pi}{4.5T}$$

$$= \frac{253.866}{\text{TICKS}}$$

where:

T = scan period in seconds
= number of CPU clock ticks times tick rate

2^{-18} = 3.814697266E-06

Refer to section 51.0, Special Coefficient Restrictions, for further restrictions.

43.2 LAG ωlg Limitations

ωlg is equal to or greater than 0.00137573 divided by TICKS.
ωlg is equal to or less than 256.055 divided by TICKS.

Low Limit $= C(2^{-18})$

$$= \frac{\omega m(2^{-18})}{TAN\left(\frac{\omega m * T}{2}\right)}$$

High Limit = 0.71C

$$= \frac{0.71\omega m}{TAN\left(\frac{\omega m * T}{2}\right)}$$

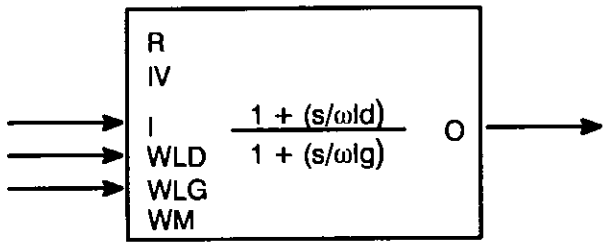
If ωm is defaulted (Wm = Ws ÷ 20):

$$\text{Low Limit} = (2^{-18})\ \frac{1.9835}{T}$$

$$= \frac{0.00137573}{TICKS}$$

44.0 LEAD/LAG

This function can be used in AutoMax Control Block tasks and UDC Control Block tasks.



Function

LAPLACE TRANSFER FUNCTION = $\frac{1 + (s/\omega ld)}{1 + (s/\omega lg)}$

Program Statement

```
CALL LEAD_LAG(INPUT = input%,                    &
    WLD = wld,                                   &
    WLG = wlg,                                   &
    WM = nnn.n,                                  &
    INITIAL_VALUE = initial_value%,              &
    RESET = reset@,                              &
    OUTPUT = output%)
```

Inputs

R (RESET) =

BOOLEAN device reset. The default for this parameter is FALSE. When this parameter is TRUE, OUTPUT will be held at INITIAL_VALUE.

IV (INITIAL_VALUE)=

INTEGER initial value. The default for this parameter is zero.

I (INPUT) =

INTEGER signal input. This parameter must be specified. It must be specified as a variable name only (literal value not accepted).

WLD (ωld) =

REAL lead frequency in radians/second. This parameter must specified. You must include a decimal point in the actual value.

WLG (ωlg) =

REAL lag frequency in radians/second. This parameter must be specified. You must include a decimal point in the actual value.

WM (ωm) =

REAL mapping frequency in radians/second. If specified, this parameter must be entered explicitly as a real literal. The default value for this parameter is ωs divided by 20 where ωs is the frequency in rad/sec. You must include a decimal point in the actual value.

Outputs

O (OUTPUT) =

INTEGER signal output. This parameter must be specified.

44.1 LEAD_LAG, ωld, ωlg, and ωm Limitations

ωld, ωlg, and ωm are equal to or greater than 0.001.

ωld, ωlg, and ωm are equal to or less than 2π divided by 4.5 T.

Low Limit = 0.001

High Limit = $\frac{2\pi}{4.5T}$

where:
T = scan period in seconds
= number of CPU clock ticks times tick rate

LEAD_LAG ωld:ωlg or ωlg:ωld ratio restrictions
maximum = 20:1
minimum = 2:1

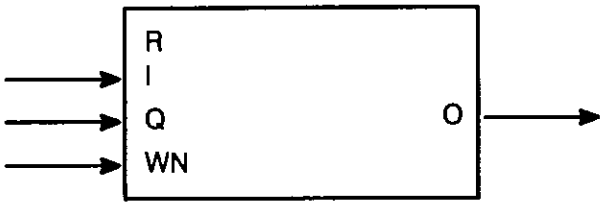
LEAD_LAG ωs:ωld and ωs:ωlg ratio restrictions
minimum = 4.5:1

where:
ωs = scan frequency in radians/second
= $\frac{2\pi}{T}$

T = scan period in seconds
= number of CPU clock ticks times tick rate

45.0 NOTCH FILTER

This function can be used in AutoMax Control Block tasks only. It cannot be used in UDC Control Block tasks.



Function

LAPLACE TRANSFER FUNCTION = $\frac{s^{**}2 + \omega n^{**}2}{s^{**}2 + \omega ns/Q + \omega n^{**}2}$

Program Statement

```
CALL NOTCH(INPUT = input%,                    &
    Q_FACTOR = q_factor,                      &
    WN = ωn,                                  &
    RESET = reset@,                           &
    OUTPUT = output%)
```

Inputs

I (INPUT) =

INTEGER signal input. This parameter must be specified as a variable name only (literal value not accepted).

Q (Q_FACTOR) =

REAL filter Q factor. Equal to 1/ (2*damping factor). May vary from .5 to 100. This parameter must be specified. You must include a decimal point in the actual value.

WN (ωn) =

REAL tunable notch filter frequency in radians/sec. May vary from .01 to 2π / 10T where T = scan time in seconds. This parameter must be specified. You must include a decimal point in the actual value.

R (RESET) =

BOOLEAN device reset. The default for this parameter is FALSE. When this parameter is TRUE, OUTPUT will be held at zero.

Outputs

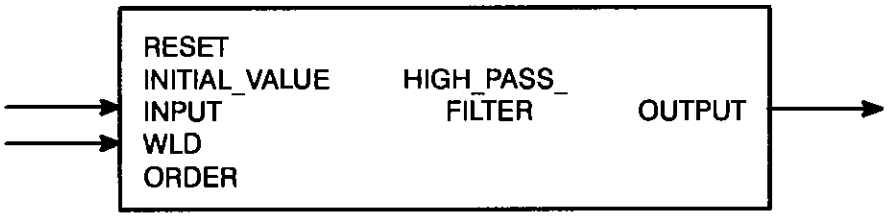
O (OUTPUT) =

INTEGER signal output. This parameter must be specified.

*Note that this block is supported only in Version 2.0 and later Programming Executive software.

46.0 HIGH-PASS FILTER (Nth Order High-Pass Butterworth Filter)

This function can be used in UDC Control Block tasks only. It cannot be used in AutoMax Control Block tasks.



Function

This function provides a high-pass filter to attenuate input frequencies that are below the cutoff frequency in UDC tasks. The order parameter can be used to change the sharpness of the cutoff.

For ORDER=1, LAPLACE TRANSFER FUNCTION = $\frac{s}{s + \omega}$

For ORDER=2, LAPLACE TRANSFER FUNCTION = $\frac{s^{**}2}{s^{**}2 + \sqrt{2}\, s\omega + \omega^{**}2}$

For ORDER=3, LAPLACE TRANSFER FUNCTION = $\frac{s^{**}3}{s^{**}3 + (2s^{**}2 * \omega) + 2s\omega^{**}2 + \omega^{**}3}$

Program Statement

```
CALL HIGH_PASS_FILTER( INPUT = input%,                    &
    WLD = ωld,                                            &
    ORDER = 1, 2, or 3                                    &
    INITIAL_VALUE = initial_value%,                       &
    RESET = reset@,                                       &
    OUTPUT = output% )
```

Inputs

RESET =

BOOLEAN output reset. The default for this parameter is FALSE. This parameter will hold OUTPUT to INITIAL_VALUE when TRUE.

INITIAL_VALUE=

INTEGER initial value. The default for this parameter is zero. When RESET = TRUE, OUTPUT will equal INITIAL_VALUE.

INPUT =

INTEGER signal input. This parameter must be specified. There is no default.

WLD =

REAL lead frequency in radians/second. This parameter must be specified. You must include a decimal point in the actual value.

ORDER =

Order of the filter transfer function. The default for this parameter is 1. If you specify this parameter, it must be a literal value (1, 2, or 3).

Outputs

OUTPUT =

INTEGER signal output. This parameter must be specified.

46.1 HIGH_PASS_FILTER ωld Limitations

ωld low limit depends on the order.
ωld must be equal to or less than 0.999π divided by T.

	1st Order	2nd Order	3rd Order
Low Limit =	$\frac{0.000004}{T}$	$\frac{.02}{T}$	$\frac{.1}{T}$

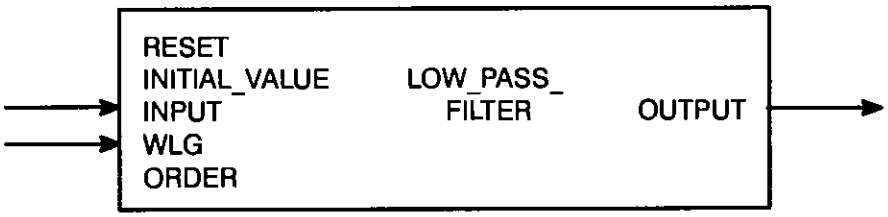
High Limit = $\frac{0.999\pi}{T}$

where:

T = scan period in seconds
= number of CPU clock ticks times 0.0005 seconds

47.0 LOW_PASS_FILTER (Nth Order Low-Pass Butterworth Filter)

This function can be used in UDC Control Block tasks only. It cannot be used in AutoMax Control Block tasks.



Function

This function provides a low-pass filter to attenuate input frequencies that are above the cutoff frequency in UDC tasks. The order parameter can be used to change the sharpness of the cutoff.

For ORDER=1, LAPLACE TRANSFER FUNCTION = $\frac{\omega}{s + \omega}$

For ORDER=2, LAPLACE TRANSFER FUNCTION = $\frac{\omega^{**2}}{s^{**2} + \sqrt{2}\, s\omega + \omega^{**2}}$

For ORDER=3, LAPLACE TRANSFER FUNCTION = $\frac{\omega^{**3}}{s^{**3} + (2s^{**2} * \omega) + 2\, s\omega^{**2} + \omega^{**3}}$

Program Statement

```
CALL LOW_PASS_FILTER( INPUT = input%,                    &
    WLG = ωlg,                                           &
    ORDER = 1, 2, or 3                                   &
    INITIAL_VALUE = initial_value%,                      &
    RESET = reset@,                                      &
    OUTPUT = output% )
```

Inputs

RESET =

BOOLEAN output reset. The default for this parameter is FALSE. This parameter will hold OUTPUT to INITIAL_VALUE when TRUE.

INITIAL_VALUE =

INTEGER initial value. The default for this parameter is zero. When RESET = TRUE, OUTPUT will equal INITIAL_VALUE.

INPUT =

INTEGER signal input. This parameter must be specified. There is no default.

WLG =

REAL lag frequency in radians/second. This parameter must be specified. You must include a decimal point in the actual value.

ORDER =

Order of the filter transfer function. The default for this parameter is 1. If you specify this parameter, it must be a literal value (1, 2, or 3).

Outputs

OUTPUT =

INTEGER signal output. This parameter must be specified.

47.1 LOW_PASS_FILTER ωlg Limitations

ωlg low limit depends upon the order.
ωlg must be equal to or less than 0.999c divided by T.

	1st Order	2nd Order	3rd Order
Low Limit =	$\frac{0.000004}{T}$	$\frac{.02}{T}$	$\frac{.1}{T}$

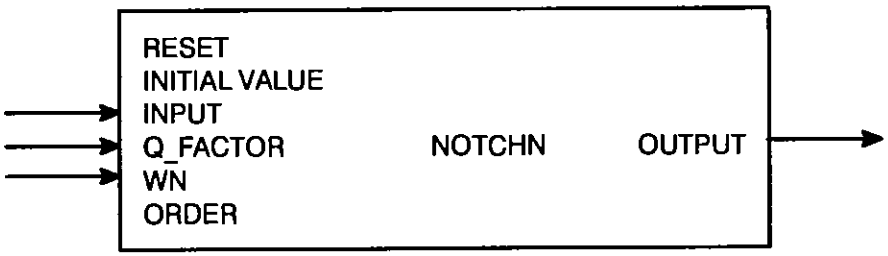
High Limit = $\frac{0.999\pi}{T}$

where:

T = scan period in seconds
= number of CPU clock ticks times 0.0005 seconds

48.0 NOTCHN (Nth Order Notch Filter)

This function can be used in UDC Control Block tasks only. It cannot be used in AutoMax Control Block tasks.



Function

This function provides a notch filter to attenuate input frequencies that are at the notch frequency in UDC tasks. The order parameter can be used to change the sharpness of the notch. The width and depth ratio of the notch can be affected by the Q factor.

$$\text{LAPLACE TRANSFER FUNCTION} = \frac{(s^{**}2 + \omega^{**}2)^{**}i}{(s^{**}2 + s\omega/Q + \omega^{**}2)^{**}i}$$

note: for ORDER = 2, 4, i will be 1, 2

Program Statement

```
CALL NOTCHN( INPUT = input%,                    &
    Q_FACTOR = q_factor%,                       &
    WN = wn,                                    &
    ORDER = 2 or 4                              &
    INITIAL_VALUE = initial_value%,             &
    RESET = reset@,                             &
    OUTPUT = output% )
```

Inputs

RESET =

BOOLEAN output reset. The default for this parameter is FALSE. This parameter will hold OUTPUT to INITIAL_VALUE when TRUE.

INITIAL_VALUE =

INTEGER initial value. The default for this parameter is zero. When RESET = TRUE, OUTPUT will equal INITIAL_VALUE.

INPUT =

INTEGER signal input. This parameter must be specified. There is no default.

Q_FACTOR =

REAL filter Q factor. Equal to 1/(2 * damping factor). This parameter must be specified. The range is 0.5 to 100.0. You must include a decimal point in the actual value.

WN =

REAL notch filter center frequency in radians/second. This parameter must be specified. You must include a decimal point in the actual value.

ORDER =

Order of the filter transfer function. The default for this parameter is 2. If you specify this parameter, it must be a literal value (2 or 4).

Outputs

OUTPUT =
INTEGER signal output. This parameter must be specified.

48.1 NOTCHN ωn Limitations

ωn low limit depends on the order.
ωn must be equal to or less than 0.999c divided by T.

$$\text{Low Limit} = \begin{array}{cc} \text{2nd Order} & \text{4th Order} \\ \dfrac{.02}{T} & \dfrac{.3}{T} \end{array}$$

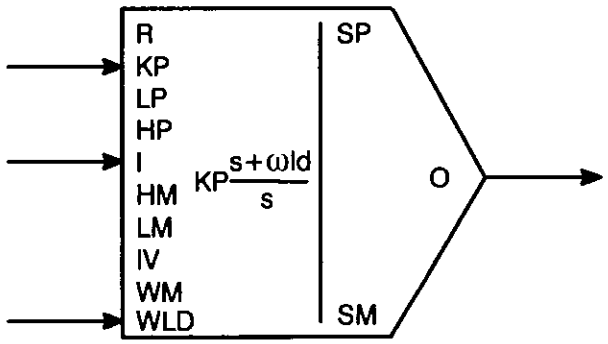
$$\text{High Limit} = \frac{0.999\pi}{T}$$

where:

T = scan period in seconds
= number of CPU clock ticks times 0.0005 seconds

49.0 PROPORTIONAL + INTEGRAL

This function can be used in AutoMax Control Block tasks and UDC Control Block tasks.



Function

$$\text{LAPLACE TRANSFER FUNCTION} = KP\left(\frac{s + \omega ld}{s}\right)$$

Program Statement

```
CALL PROP_INT(INPUT = input%,                          &
     KP = Kp,                                          &
     WLD = wld,                                        &
     WM = nnn.n,                                       &
     INITIAL_VALUE = initial_value%,                   &
     LIMIT_PLUS = limit_plus%,                         &
     LIMIT_MINUS = limit_minus%,                       &
     RESET = reset@,                                   &
     HOLD_PLUS = hold_plus@,                           &
     HOLD_MINUS = hold_minus@,                         &
     SATURATED_PLUS = saturated_plus@,                 &
     SATURATED_MINUS = saturated_minus@,               &
     OUTPUT = output%)
```

Inputs

R (RESET) =

BOOLEAN integrator reset. The default for this parameter is FALSE. When this parameter is TRUE, OUTPUT will be held at INITIAL_VALUE.

KP (KP) =

REAL proportional gain. This parameter must be specified. You must include a decimal point in the actual value.

LP (LIMIT_PLUS) =

INTEGER integrator upper limit. The default for this parameter is 32767. This parameter will limit OUTPUT from becoming more positive. It will not prevent the output value from becoming more negative.

HP (HOLD_PLUS) =

BOOLEAN integrator hold plus. The default for this parameter is FALSE. When this parameter is TRUE, INTEGRATOR will be prevented from beocming more positive. It will be permitted to go more negative. If KP changes, the output will change even if HOLD_PLUS is TRUE.

I (INPUT) =

INTEGER signal input. This parameter must be specified.

HM (HOLD_MINUS) =

BOOLEAN integrator hold minus. The default for this parameter is FALSE. When this parameter is TRUE, INTEGRATOR will be prevented from becoming more negative. It will be permitted to go more positive. If KP changes, the output will change even if HOLD_MINUS is TRUE.

LM (LIMIT_MINUS) =

INTEGER integrator lower limit. The default for this parameter is −32768. This parameter will limit OUTPUT from becoming more negative. It will not prevent the output value from becoming more positive.

IV (INITIAL_VALUE) =

INTEGER initial value of integrator. The default for this parameter is zero.

WM (ωm) =

REAL mapping frequency in radians/second. If specified, this parameter must be entered explicitly as a real literal. The default value for this parameter is ωs divided by 20. You must include a decimal point in the actual value.

WLD (ωld) =

REAL lead frequency. This parameter must be specified. You must include a decimal point in the actual value.

Outputs

SP (SATURATED_PLUS) =

BOOLEAN SATURATED plus output. This parameter is optional. TRUE if OUTPUT% reaches LIMIT(+).

O (OUTPUT) =

INTEGER signal output. This parameter must be specified.

SM (SATURATED_MINUS) =

BOOLEAN saturated minus output. This parameter is optional. TRUE if OUTPUT% reaches LIMIT(−).

49.1 PROP_INT ωm Limitations

ωm is equal to or greater than 2^{-17} divided by T. ωm is equal to or less than .9π divided by T.

$$\text{Low Limit} = \left(\frac{2}{T}\right) \text{ARCTAN}\,(2^{-18})$$

$$= \left(\frac{2\,(2^{-18})}{T}\right)$$

$$= \frac{2^{-17}}{T}$$

$$\text{High Limit} = \frac{.9\pi}{T}$$

where:

T = scan period in seconds
= number of CPU clock ticks times tick rate

Refer to section 51.0, Special Coefficient Restrictions, for further restrictions.

49.2 PROP_INT ωld Limitations

ωld is equal to or greater than C times 2^{-18} divided by KP. ωld is equal to or less than .9π divided by T.

$$\text{Low Limit} = \frac{C(2^{-18})}{KP}$$

$$= \left(\frac{2^{-18}}{KP}\right)\left(\frac{Wm}{\text{TAN}\left(\frac{Wm*T}{2}\right)}\right)$$

$$\text{High Limit} = \frac{.9\pi}{T}$$

If ωm is defaulted (ωm = ωs ÷ 20):

$$\text{Low Limit} = \left(\frac{2^{-18}}{KP}\right)\left(\frac{1.9835}{T}\right)$$

where:
C = bilinear mapping constant

$$= \frac{\omega m}{TAN\left(\frac{\omega m * T}{2}\right)}$$

2^{-18} = 3.814697266E-06
T = scan period in seconds
= number of CPU clock ticks times tick rate
ωs = scan frequency in radians/second

$$= \frac{2\pi}{T}$$

Refer to section 51.0 Special Coefficient Restrictions, for further restriction notes.

49.3 PROP_INT KP Limitations

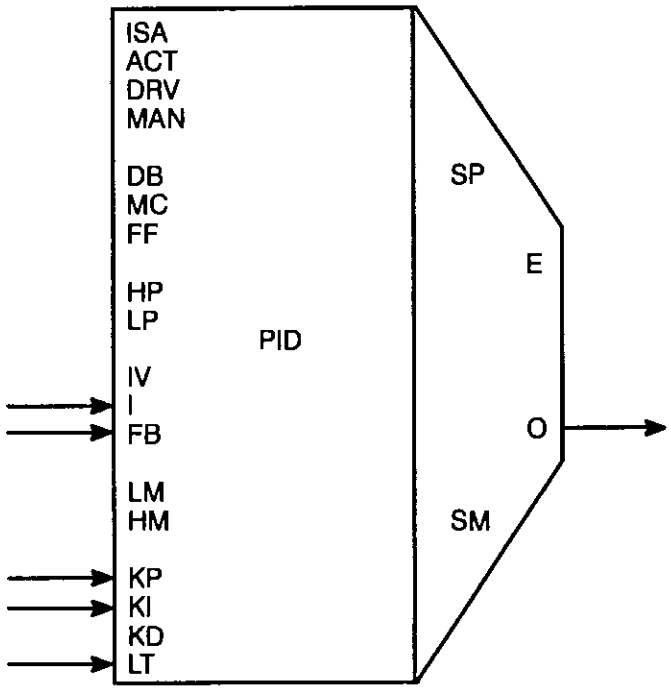
KP is equal to or greater than 0.001. KP is equal to or less than 128.0.

Low Limit = .001
High Limit = 128.0

Refer to section 51.0 Special Coefficient Restrictions, for further restriction notes.

50.0 PID

This function can be used in AutoMax Control Block tasks only. It cannot be used in UDC Control Block tasks.



Function

If MANUAL is true then

ERROR = 0
INTEGRATOR = INITIAL_VALUE
SUM = INTEGRATOR

Else

(automatic mode)
If REVERSE_ACTION is true then
ERROR = FEEDBACK - INPUT

Else

ERROR - INPUT - FEEDBACK
P_TERM = ERROR * KP
I_TERM = (ERROR + ERROR (n-1)) * KI * LOOP_TIME / 2
If abs (I_TERM) < DEAD_BAND then
I_TERM = 0
If I_TERM > 0 and HOLD_PLUS is true then
I_TERM = 0
If I_TERM < 0 and HOLD_MINUS is true then
I_TERM = 0
INTEGRATOR = INTEGRATOR + I_TERM
If INTEGRATOR > LIMIT_PLUS then
INTEGRATOR = LIMIT_PLUS
If INTEGRATOR < LIMIT_MINUS then
INTEGRATOR = LIMIT_MINUS

```
If DERIVATIVE = TRUE then
      IF ACTION = FALSE
           D_ERROR = -FEEDBACK
      Else D_ERROR = FEEDBACK
      End If
Else
      If ACTION = FALSE then
      D_ERROR = INPUT - FEEDBACK
      Else
      D_ERROR = FEEDBACK - INPUT
      End If
End If
Else
      D_ERROR = ERROR
D_TERM = (D_ERROR - D_ERROR (n-1) ) * KD
SUM = INTEGRATOR + P_TERM + D_TERM

(evaluated if in manual or automatic mode)
  SUM = SUM + FEED_FORWARD
  CHANGE = SUM-OUTPUT (n-1)
  If CHANGE > MAX_CHANGE then
  CHANGE = MAX_CHANGE
  Else if CHANGE < MAX_CHANGE then
  CHANGE = -MAX_CHANGE
  OUTPUT = OUTPUT + CHANGE

  If OUTPUT >= LIMIT_PLUS then
  OUTPUT = LIMIT_PLUS, SATURATED_PLUS = 1
Else if OUTPUT <= LIMIT_MINUS then
OUTPUT = LIMIT_MINUS, SATURATED_MINUS = 1
Else
SATURATED_PLUS = 0, SATURATED_MINUS = 0
  Note: If ISA is true then KI = KI * KP and KD = KD * KP.
```

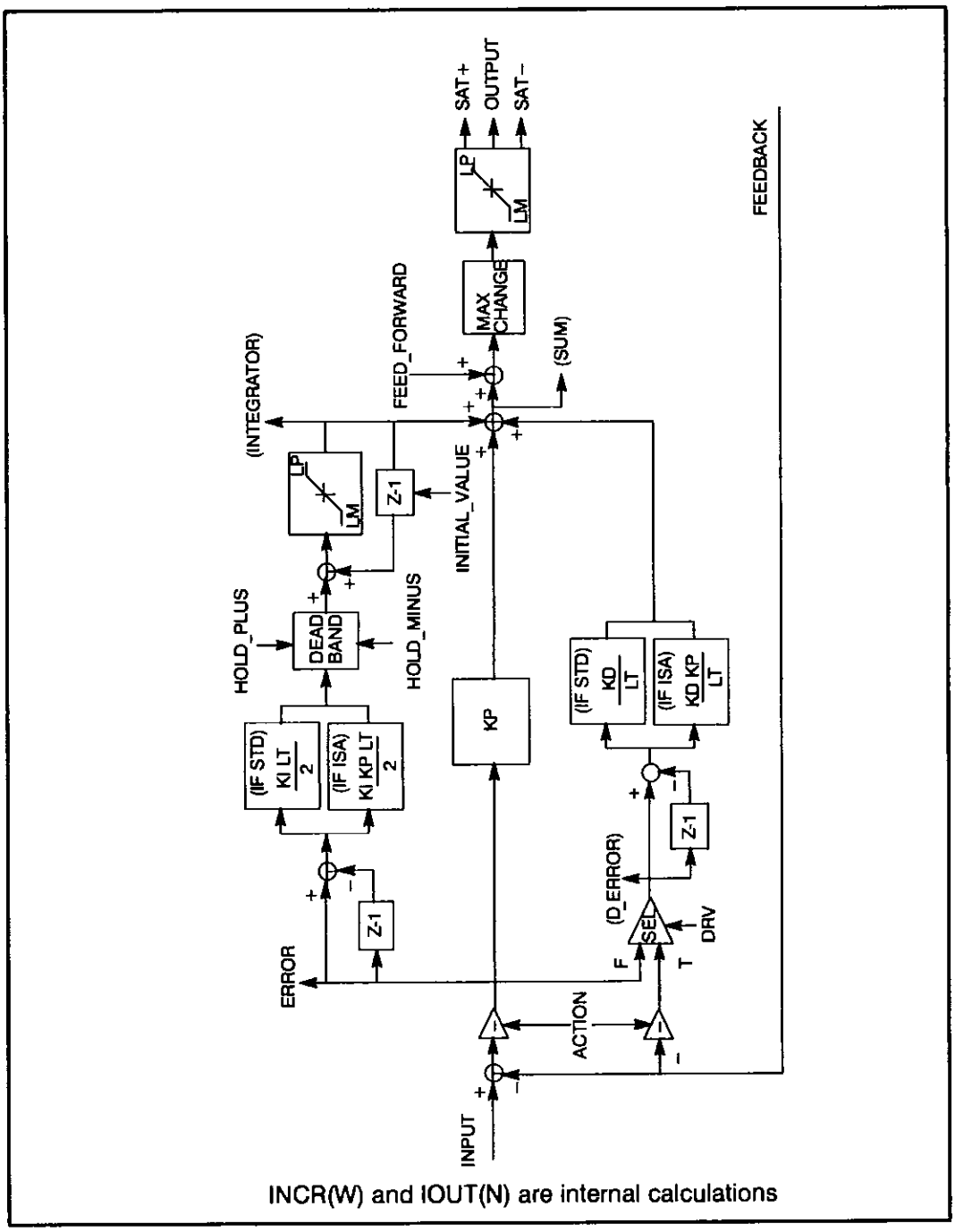


Figure 50.1 - PID Block Diagram

Program Statement

```
CALL PID(ISA=<boolean literal>,                    &
    ACTION=<boolean literal>,                      &
    DERIVATIVE=<boolean literal>,                  &
    MANUAL=manual@,                                &
    KP=kp,                                         &
    KI=ki,                                         &
    KD=kd,                                         &
    LOOP_TIME=loop_time,                           &
    INITIAL_VALUE=initial_value%,                  &
```

```
FEED_FORWARD=feed_forward%,                    &
INPUT=input%,                                  &
FEEDBACK=feedback%,                            &
DEAD_BAND=dead_band%,                          &
MAX_CHANGE=max_change%,                        &
LIMIT_PLUS=limit_plus%,                        &
LIMIT_MINUS=limit_minus%,                      &
HOLD_PLUS=hold_plus@,                          &
HOLD_MINUS=hold_minus@,                        &
SATURATED_PLUS=saturated_plus@,                &
SATURATED_MINUS=saturated_minus@,              &
ERROR=error%,                                  &
OUTPUT=output%)
```

Inputs

ISA (ISA) =

This input selects if KI and KD are scaled by KP. If this optional input is programmed, it must be a boolean literal. If it is not programmed, it defaults to a value of FALSE.
If ISA = TRUE then the KI and KD coefficients are automatically multiplied by KP.
If ISA = FALSE then KI and KD are independent of KP.

ACTION (ACTION) =

This input selects how ERROR is calculated. If this optional input is programmed, it must be a boolean literal. If it is not programmed, it defaults to a value of FALSE.
If ACTION = TRUE then ERROR = FEEDBACK - INPUT.
If ACTION = FALSE then ERROR = INPUT - FEEDBACK.

DRV (DERIVATIVE) =

This input selects which signal to use in calculating the derivative term. If this optional input is programmed, it must be a boolean literal. If it is not programmed, it defaults to a value of FALSE.

```
If DERIVATIVE = TRUE then
     IF ACTION = FALSE
          D_ERROR = -FEEDBACK
     Else D_ERROR = FEEDBACK
     End If
Else
     If ACTION = FALSE then
     D_ERROR = INPUT - FEEDBACK
     Else
     D_ERROR = FEEDBACK - INPUT
     End If
End If
```

MAN (MANUAL) =

This input selects between manual and automatic mode of operation. If this optional input is programmed, it must be a boolean variable or literal. If it is not programmed, it defaults to a value of FALSE.

If MANUAL = TRUE then the PID block is in manual mode. Internal variables are reset to zero and the output is the sum of the INITIAL_VALUE and FEED_FORWARD.

If MANUAL = FALSE then the PID block is in automatic mode and the selected algorithm is calculated.

KP (KP) =

This input is the gain on the proportional term. It must be programmed as a real variable or constant. A decimal point must be included in a constant value. The value of KP is dimensionless.

KI (KI) =

This input is the gain on the integral term. It must be programmed as a real variable or constant. A decimal point must be included in a constant value. The units on KI are repeats per second (I/seconds).

KD (KD) =

This input is the gain on the derivative term. If this optional input is programmed, it must be programmed as a real variable or constant. A decimal point must be included in a constant value. The units on KD are seconds.

LT (LOOP_TIME) =

This input is used to tell the PID block how often it is being executed. This parameter must be specified as a constant. A variable name is not accepted. A decimal point must be included in a constant value. The units are in seconds. If the PID block is evaluated on every scan of the task, this value is the number of ticks per scan times the tick rate. If the PID block is not evaluated on every scan, this value is the time in seconds between iterations. The PID block must be called on a consistent time basis.

IV (INITIAL_VALUE) =

This input is used to define an initial value for the integrator. If this optional input is programmed, it must be an integer variable or constant. If not programmed, it defaults to a value of zero. When the PID block is in manual mode, this value is loaded into the integrator.

FF (FEED_FORWARD) -

This input is used to define a feed forward variable. If this optional input is programmed, it must be an integer variable or constant. If not programmed, it defaults to a value of zero. FEED_FORWARD is added to SUM to produce OUTPUT.

I (INPUT) =

This input is used to define the set point signal to the PID block. This parameter must be programmed as an integer variable. A constant is not accepted.

FB (FEEDBACK) =

This input is used to define the feedback signal to the PID block. This parameter must be programmed as an integer variable. A constant is not accepted.

DB (DEAD_BAND) =

This input is used to define a deadband on integral calculation. If this optional input is programmed, it must be an integer variable or constant. If not programmed, it defaults to a value of zero. If the absolute value of ERROR is less than DEAD_BAND, no change is made to the value of the integrator.

MC (MAX_CHANGE) =

This input is used to define a maximum rate of change of OUTPUT. If this optional input is programmed, it must be an integer variable or constant. If not programmed, it defaults to a value of 32767.

LP (LIMIT_PLUS) =

This input is used to define a positive limit on the INTEGRATOR and on OUTPUT. If this optional input is programmed, it must be an integer variable or constant. If not programmed, it defaults to a value of 32767. The INTEGRATOR is limited to never be greater than this value. The OUTPUT is limited to never be greater than this value.

LM (LIMIT_MINUS) =

This input is used to define a minus limit on the INTEGRATOR and on OUTPUT. If this optional input is programmed, it must be an integer variable or constant. If not programmed, it defaults to a value of −32767. The INTEGRATOR is limited to never be less than this value. The OUTPUT is limited to never be less than this value.

HP (HOLD_PLUS) =

This input is used to hold increasing the INTEGRATOR. If this optional input is programmed, it must be a boolean variable. If not programmed, it defaults to a value of FALSE. If ERROR > 0 and HOLD_PLUS is true, the INTEGRATOR is not allowed to become more positive.

HM (HOLD_MINUS) =

This input is used to hold decreasing the INTEGRATOR. If this optional input is programmed, it must be a boolean variable. If not programmed, it defaults to a value of FALSE. If ERROR < 0 and HOLD_MINUS is true, the INTEGRATOR is not allowed to become more negative.

Outputs

SP (SATURATED_PLUS) =

This output is used to indicate that OUTPUT is being limited by LIMIT_PLUS. If this optional output is programmed, it must be a boolean variable. If the calculated value for OUTPUT would be >= LIMIT_PLUS, this output is turned on. Otherwise, it is turned off.

SM (SATURATED_MINUS) =

This output is used to indicate that OUTPUT is being limited by LIMIT_MINUS. If this optional input is programmed, it must be a boolean variable. If the calculated value for OUTPUT would be <= LIMIT_MINUS, this output is turned on. Otherwise, it is turned off.

E (ERROR) =

This output is used to define the variable where ERROR can be displayed. If this optional output is programmed, it must be an integer variable. The value displayed will be INPUT – FEEDBACK or FEEDBACK - INPUT as defined by the ACTION input.

O (OUTPUT) =

This output is used to define the variable where the resulting calculation will be written. This output is required, and must be defined as an integer variable. When the PID block is in manual mode, OUTPUT is the sum of INITIAL_VALUE and FEED_FORWARD. When the PID block is in automatic mode, it is the sum of the KP, KI, and KD terms, and FEED_FORWARD.

50.1 KP Limitations

The value of KP is dimensionless. KP is limited to values between the following limits:

Low limit = 0
High limit = 9.2233717 x 10**18

The smallest value KP can represent (other than 0) is 5.4210107 x 10**(-20).

When KP is programmed as a symbol, values less than zero are clamped at zero. When KP is programmed as a literal, values less than zero will cause a compiler error.

50.2 KI Limitations

KI units are 1/seconds. KI is limited to values between the following limits:

Low limit = 0
High limit = 9.2233717 x 10**18

The smallest value KI can represent (other than 0) is 5.4210107 x 10**(-20).

When KI is programmed as a symbol, values less than zero are clamped at zero. When KI is programmed as a literal, values less than zero will cause a compiler error.

50.3 KD Limitations

KD units are seconds. KD is limited to the following values:

Low limit = 0
High limit = 9.2233717 x 10**18

The smallest value KD can represent (other than 0) is 5.4210107 x 10**(-20).

When KD is programmed as a symbol, values less than zero are clamped at zero. When KD is programmed as a literal, values less than zero will cause a compiler error.

50.4 LOOP_TIME Limitations

LOOP_TIME units are seconds (the time between updates). LOOP_TIME is limited to the following values:

Low limit = .01 or 1.0 x 10**(-2) seconds
High limit = 9.2233717 x 10**18 seconds

Values less than .01 will cause a compiler error.

50.5 DEAD_BAND Limitations

DEAD_BAND is limited to the following values:

Low limit = 0
High limit = 32767

When DEAD_BAND is programmed as a symbol, values less than zero are clamped at zero. When DEAD_BAND is programmed as a literal, values less than zero will cause a compiler error.

50.6 MAX_CHANGE Limitations

When MAX_CHANGE is programmed as a symbol, it is limited to the following values:

Low limit = 0
High limit = 32767

Values less than zero are clamped at zero.

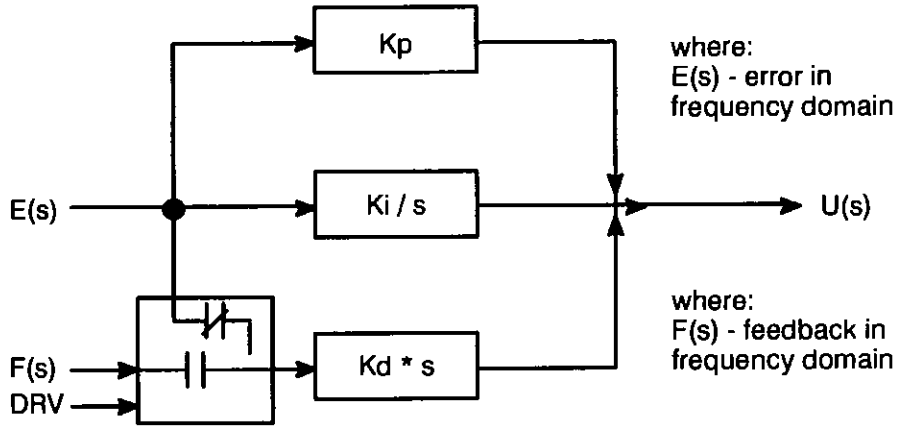
When MAX_CHANGE is programmed as a literal, it is limited to the following values:

Low limit = 1
High limit = 32767

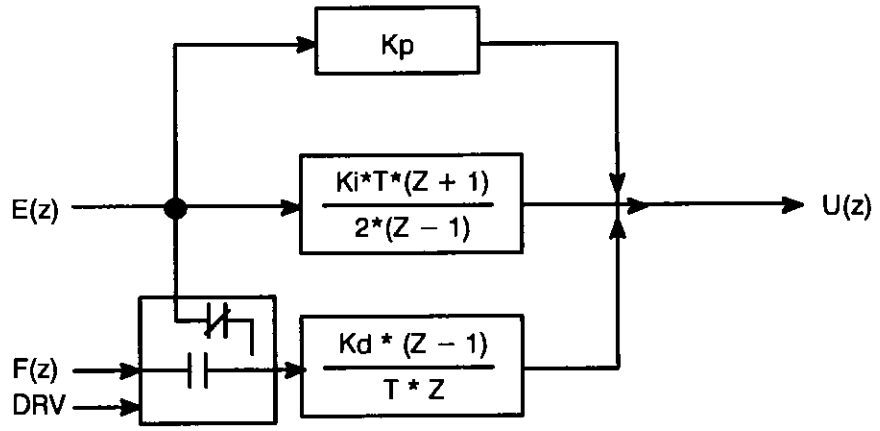
Values less than one will cause a compiler error.

Independent Mode: Derivative on Error or Feedback (n)

s - domain functional block:



Z - domain functional block:



where:
E(z) - error in z-domain
F(z) - feedback in z-domain

Difference Equation:

Error(n) = Input(n) - Feedback(n) if action = FALSE or
Feedback(n) - Input(n) if action = TRUE

Dev_err(n) = Error(n) if derivative = FALSE or
Feedback(n) if derivative = TRUE

Incr(n) =

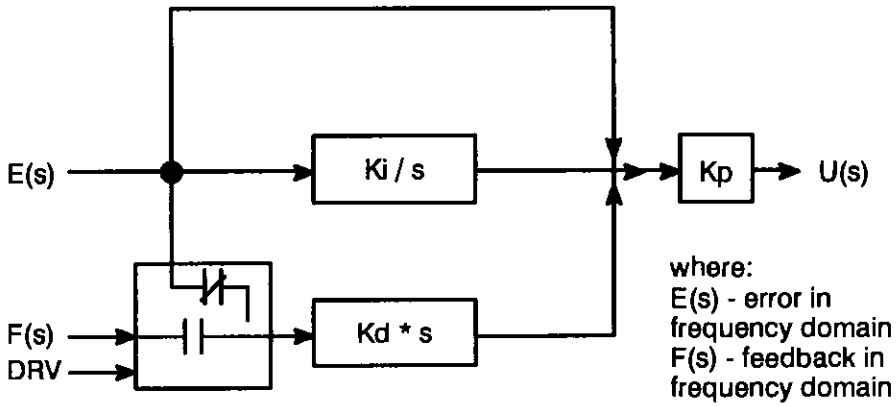
(P term) Kp * [Error(n) - Error(n − 1)]
(I term) + K1 * [Error(n) + Error(n − 1)]
(D term) + K2 * [Dev_err(n) − 2*Dev_err(n−1) + Dev_err(n−2)]
+ Incr_remainder(n−1)

Output(n) = Output(n−1) + Incr(n)

where: $K1 = \frac{Ki * T}{2}$ $K2 = \frac{Kd}{T}$

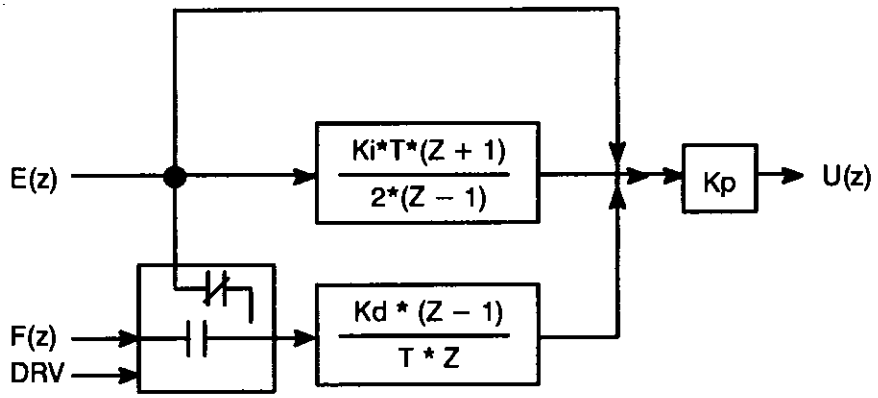
ISA Mode: Derivative on Error(n) or Feedback (n)

s - domain functional block:



where:
E(s) - error in frequency domain
F(s) - feedback in frequency domain

Z - domain functional block:



where:
E(z) - error in z-domain
F(z) - feedback in z-domain

Difference Equation:

Error(n) = Input(n) - Feedback(n) if action = FALSE or
Feedback(n) - Input(n) if action = TRUE

Dev_err(n) = Error(n) if derivative = FALSE or
Feedback(n) if derivative = TRUE

Incr(n) =
Kp * ([Error(n) - Error(n − 1)]
(I term) + K1 * [Error(n) + Error(n − 1)]
(D term) + K2 * [Dev_err(n) - 2*Dev_err(n−1) + Dev_err(n−2)]
+ Incr_remainder(n−1)

Output(n) = Output(n−1) + Incr(n)

where: $K1 = \frac{Ki * T}{2}$ $K2 = \frac{Kd}{T}$

51.0 SPECIAL COEFFICIENT RESTRICTIONS

This section is relevant to the following control blocks: LAG, DIFF_LAG, INTEGRATE, and PROP_INT.

The input signal to these control blocks is in the range −32768 to +32767. The output signal, however, may be limited to something less than this range. The cause of this output range limitation is a trade-off between the range and resolution of the internal math used. As the coefficient value decreases in size, resolution is retained at the expense of range. In the majority of cases, this should not pose a problem due to the normalization factor of 1PN = 4095.

These blocks contain an integrator. As the integrator coefficient value decreases, its output value may be limited to less than a 16-bit value in order to retain resolution and still use 32-bit math. The following shows the relationship between coefficient value and output limitation.

Coefficient Value	Max Output Range
2^{-18} to .124996	±8191
.125 to .249992	±16383
.25 to max_val	±32767

The integrator coefficient for each block can be calculated as follows:

LAG: $Kx = \frac{\omega lg}{C}$

DIFF_LAG: $Kx = \frac{\omega lg}{C}$

INTEGRATE: $Kx = \frac{KI}{C}$

PROP_INT: $Kx = \frac{\omega ld \times Kp}{C}$

where:

$$C = \frac{\omega m}{TAN\left(\frac{\omega m}{2} * T\right)}$$

T = scan period in seconds.

If ωm is defaulted (not programmed), then

$$\omega m = \frac{\omega s}{20} = \frac{2\pi}{20T} = \frac{\pi}{10T}$$

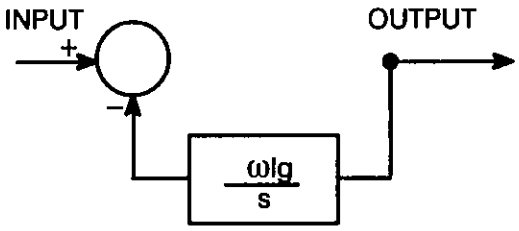
and

$$C = \frac{\left(\frac{\pi}{10T}\right)}{TAN\left(\frac{\pi}{20}\right)}$$

$$= \frac{\frac{.314159}{T}}{.158386444}$$

$$= \frac{1.9835}{T}$$

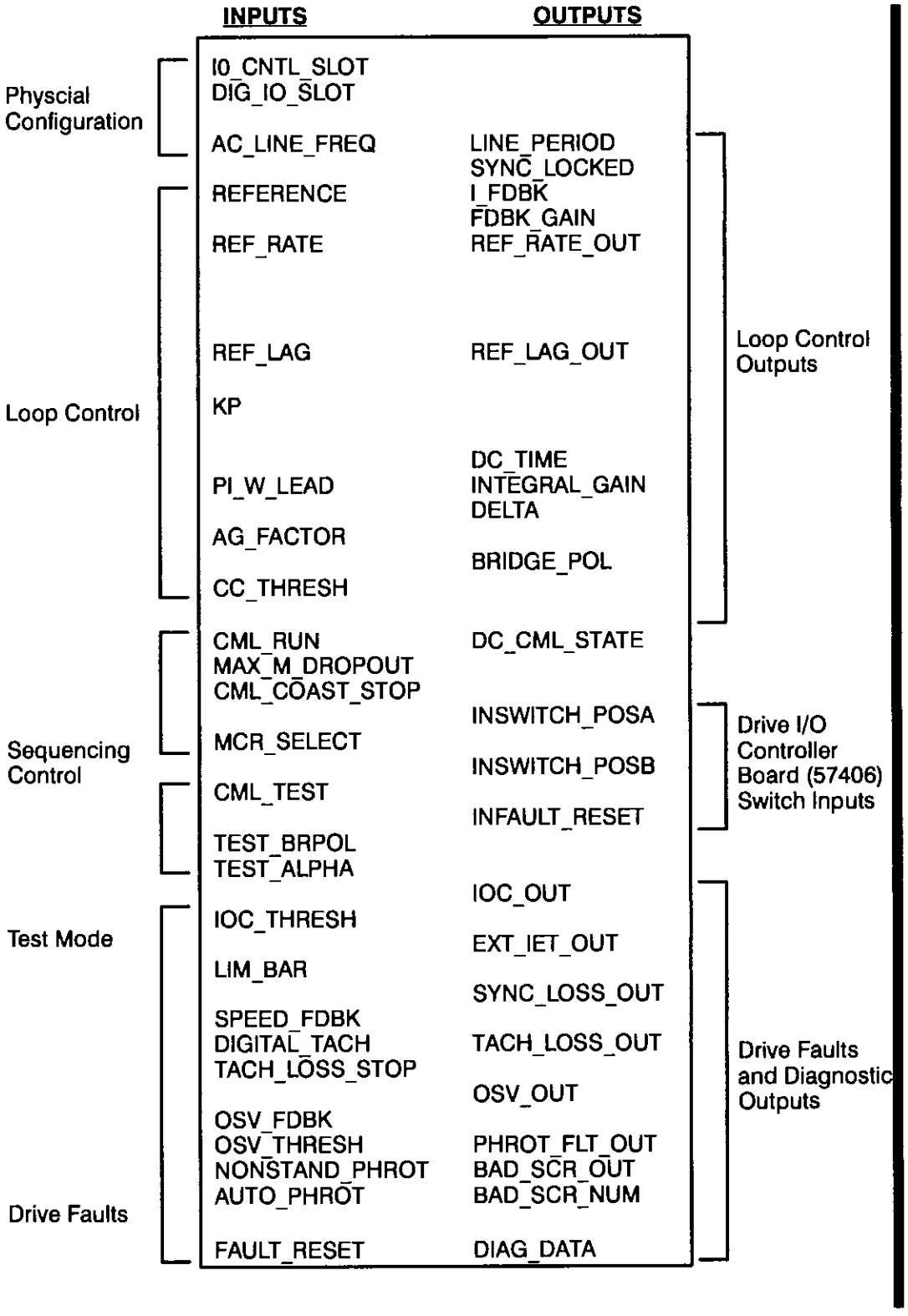
The DIFF_LAG block is a special case because the integrator is in the feedback path:



If the integrator output is limited, the OUTPUT is limited to the input minus the output of the integrator. Therefore, if the integrator output is limited to 8191 and the input is set to 32767, the output will settle out to 32767 − 8191 = 24576, not zero, as would be expected with a DIFF_LAG function. Therefore, if the coefficient value can cause the integrator output to be limited to less than ±32767, make sure that the input does not exceed the limit of the integrator.

52.0 DC DRIVE CURRENT MINOR LOOP

This function can be used in AutoMax Control Block tasks only. It cannot be used in AutoMax PC3000 Control Block tasks or UDC Control Block tasks.



PROGRAM STATEMENT:

```
CALL DC_DRIVE_CML(                          &
  IO_CNTL_SLOT = io_cntl_slot%,             &
  DIG_IO_SLOT = dig_io_slot,                &
  AC_LINE_FREQ = ac_line_freq%,             &
  REFERENCE = reference%,                   &
  REF_RATE = ref_rate,                      &
  REF_LAG = ref_lag,                        &
  KP = kp,                                  &
  PI_W_LEAD = pi_w_lead,                    &
  AG_FACTOR = ag_factor%,                   &
  CC_THRESH = cc_thresh%,                   &
  CML_RUN = cml_run@,                       &
  CML_COAST_STOP = cml_coast_stop@,         &
  MCR_SELECT = mcr_select@,                 &
  MAX_M_DROPOUT = max_m_dropout,            &
  CML_TEST = cml_test@,                     &
  TEST_BRPOL - test_brpol@,                 &
  TEST_ALPHA = test_alpha%,                 &
  IOC_THRESH = ioc_thresh%,                 &
  LIM_BAR = lim_bar%,                       &
  DIGITAL_TACH = digital_tach@,             &
  SPEED_FDBK = speed_fdbk%,                 &
  TACH_LOSS_STOP = tach_loss_stop@,         &
  OSV_FDBK = osv_fdbk%,                     &
  OSV_THRESH = osv_thresh%,                 &
  NONSTAND_PHROT = nonstand_phrot@,         &
  AUTO_PHROT = auto_phrot@,                 &
  FAULT_RESET = fault_reset%,               &
  DC_CML_STATE = dc_cml_state@,             &
  SYNC_LOCKED = sync_locked@,               &
  LINE_PERIOD = line_period%,               &
  INSWITCH_POSA = inswitch_posa@,           &
  INSWITCH_POSB = inswitch_posb@,           &
  INFAULT_RESET = infault_reset@,           &
  DC_TIME = dc_time%,                       &
  BRIDGE_POL = bridge_pol@,                 &
  REF_RATE_OUT = ref_rate_out%,             &
  REF_LAG_OUT = ref_lag_out%,               &
  FDBK_GAIN = fdbk_gain%,                   &
  I_FDBK = i_fdbk%,                         &
  INTEGRAL_GAIN = integral_gain%,           &
  DELTA = delta%,                           &
  IOC_OUT = ioc_out@,                       &
  SYNC_LOSS_OUT = sync_loss_out@,           &
  TACH_LOSS_OUT = tach_loss_out@,           &
  OSV_OUT = OSV = osv_out@,                 &
  EXT_IET_OUT = ext_iet_out@,               &
  PHROT_FLT_OUT = phrot_flt_out@,           &
  BAD_SCR_OUT = bad_scr_out@,               &
  BAD_SCR_NUM = bad_scr_num%,               &
  DIAG_DATA = diag_data% )
```

NOTE: *FORCE is supported for inputs (unless otherwise stated).*
FORCE is not supported for outputs.

Function

Performs the current minor loop regulation for S6R DC motor drives. This includes standard drive sequencing from M-contactor control to enabling of the current minor loop and all inherent drive fault detections and processing. This block can only be used for systems that include drives. A task that includes the CML block cannot include a SCAN_LOOP block.

A CML task requires a hardware interrupt line. Refer to the AutoMax Enhanced BASIC Language Instruction manual (J-3675) for information on interrupt line allocation.

The Processor running the task that contains the CML block must be set to the default tick rate (5.5ms).

52.1 Input Keyword Definitions

The phrase "fixed at task initialization" means that this input must be defined before the statement CALL DC_DRIVE_CML is executed after the task is first turned on. The value of this input at this time will be the value used until the task is turned off.

If any dynamic input is unacceptable, such as being out of range, an error will be logged. The value will not be used and the last acceptable input is retained. In such instances, the user input value may not reflect the actual value used.

When the task is initially turned on, the existing values for dynamic inputs are initialized to zero (FALSE for boolean variables) before checking the programmed value. Therefore, if the programmed value is unacceptable when the task is initially turned on, the value used will be zero, or FALSE (180 degrees in the case of TEST_ALPHA%).

Table 1 - Input Parameters Summary

Parameter Keyword	Data Type	Symbol or Literal	Dynamic 1	Range	Default
IO_CNTL_SLOT	Integer %	Literal	No	0 to 15	None
DIG_IO_SLOT	Integer %	Literal	No	0 to 15	None
AC_LINE_FREQ	Integer %	Literal	No	48 to 62	60
REFERENCE	Integer %	Symbol	Yes	−4095 to +4095	None
REF_RATE	Real	Either	Yes	0.00035 to 11.37(60Hz)	0.0
REF_LAG	Real	Either	No	1 to 500(60Hz) 1 to 400(50Hz)	200.0
KP	Real	Either	Yes	0.0 to 4.0	None
PI_W_LEAD	Real	Either	No	10.0 to 500.0	None
AG_FACTOR	Integer%	Either	No	1 to 300	1
CC_THRESH	Integer%	Either	Yes	0 to 32767	0
CML_RUN	Boolean@	Symbol	Yes	True/False	None
MAX_M_DROPOUT	Real	Either	No	0.1 to 1.0	0.1
CML_COAST_STOP	Boolean@	Symbol	Yes	True/False	False
MCR_SELECT	Boolean@	Either	Yes	True/False	False
CML_TEST	Boolean@	Either	Yes	True/False	False
TEST_BRPOL	Boolean@	Either	Yes	True/False	False (bridge#1)
TEST_ALPHA	Integer%	Either	Yes	0 to 180	180
IOC_THRESH	Integer%	Either	No	100 to 400	None
LIM_BAR	Integer%	Either	No	100 to 400	None
SPEED_FDBK	Integer%	Symbol	Yes	4095=1PN	No tach loss
DIGITAL_TACH	Boolean@	Either	No	True/False	True
TACH_LOSS_STOP	Boolean@	Either	No	True/False	True
OSV_FDBK	Integer%	Either	Yes	±32767	No OSV fault
OSV_THRESH	Integer%	Either	No	0 to 32767	0
NONSTAND_PHROT	Boolean@	Either	Yes	True/False	False
AUTO_PHROT	Boolean@	Either	No	True/False	False
FAULT_RESET	Boolean@	Symbol	Yes	True/False	None

1 Dynamic indicates whether the parameter value will be recognized if modified after the task has been turned on.
No: The value is read in only once when the task is initially turned on.
Yes: See the detailed description for that parameter to determine when the new value is read in.

52.1.1 Physical Configuration Inputs

IO_CNTL_SLOT%: The slot number of the drive I/O controller module.

- Must be a single word integer entered as a literal (variable name not accepted).
- Range = 0 to 15 in steps of 1.
- Fixed at task initialization.
- No default exists for this input; it must be entered.

DIG_IO_SLOT%: The slot number of the drive digital I/O module.

- Must be a single word integer entered as a literal (variable name not accepted).
- Range = 0 to 15 in steps of 1.
- Fixed at task initialization.
- No default exists for this input; it must be entered.

AC_LINE_FREQ%: The frequency (in hertz) of the AC line power to the drive.

- Must be a single word integer entered as a literal (variable name not accepted). If it is known to vary, the lowest expected frequency must be entered.
- Range = 48 to 62 in steps of 1.
- Fixed at task initialization.
- Default = 60 (Hz).

52.1.2 Loop Control Inputs

REFERENCE%: Symbol assigned as the reference to the current minor loop (CML).

- Must be a single word integer variable name (literal not accepted).
- A plus value turns the forward power unit on (bridge #1: 1 THY through 6 THY). A minus value turns the reverse power unit on (bridge #2: 11 THY through 16 THY).
- Range is +4095 through −4095 in steps of 1. This range limitation is not enforced by the CML itself; it must be enforced by the application task generating this signal. Exceeding this range will cause unpredictable results.
- May be modified dynamically. Latched at the start of every CML scan (nominally 1/2.7778 msec at 60 Hz AC line frequency).
- No default exists for this input; it must be entered.

REF_RATE: Symbol or value assigned as the CML reference rate limit.

- Must be a real number (variable name or literal accepted), entered as the time in seconds for a reference step change of 0 to current reference limit (LIM_BAR).
- Range = .00035 to 11.37 (60 Hz AC line frequency).

- May be modified dynamically. Latched-in once every system clock tick while the drive is running (DC_CML_STATE@ = TRUE).
- Default = 0 second (i.e., rate limit function removed)
- The internally calculated rate in counts/scan is RATE = (LIM_BAR/INPUT) * (sec/scan)
- where:
  LIM_BAR = 4095 internally (always)
  sec/scan = 1 divided by 6 times the power bridge AC line frequency

RATE is limited. RATE is equal to or greater than zero. RATE is equal to or less than 32767. Therefore, the limits on REF_RATE vary as a function of the measured AC line frequency to the DC drive power bridge.

REF_LAG: Symbol or value assigned as the CML reference lag break frequency (ωlg).

- Must be a real number in rad/sec (variable name or literal accepted).
- Range is 1 through 500 rad/sec (at 60 Hz line) and 1 through 400 rad/sec (at 50 Hz line) in steps of 1 rad/sec.
- Fixed at task initialization
- Default = 200.0 rad/sec.

KP: Symbol or value assigned as the CML proportional gain.

- Must be a real number (variable name or literal accepted).
- Range is 0 through 4.0 in steps of .0009765 (1/1024).
- May be modified dynamically. Latched-in once every system clock tick while the drive is running (DC_CML_STATE@ = TRUE).
- No default exists for this input; it must be entered.

PI_W_LEAD: Symbol or value assigned as the CML P+I lead break frequency.

- Must be a real number in rad/sec (variable name or literal accepted).
- Range = 10 through 500 rad/sec in steps of 1 rad/sec.
- Fixed at task initialization.
- No default exists for this input; it must be entered.

AG_FACTOR%: Symbol or value assigned as the CML adaptive gain factor, which is the maximum ratio between the integral gain from continuous conduction to discontinuous conduction.

- Must be a single word integer (variable name or literal accepted).
- Range = 1 through 300 in steps of 1.
- Fixed at task initialization.
- Default = 1 (non-adaptive current loop).

CC_THRESH%: Symbol or value assigned as the continuous conduction threshold value used in the adaptive gain logic.

- Must be a single word integer (variable name or literal accepted).
- Range = 0 through 32767 in steps of 1.
- May be modified dynamically.
- Default = 0.

52.1.3 Sequencing Control Inputs

CML_RUN@: Symbol assigned to the CML run/stop switch input.

- Must be a boolean variable name (literal not accepted).
- TRUE (ON) = run CML; FALSE (OFF) = stop CML.
- May be modified dynamically. Latched-in once every system clock tick.
- No default exists for this input; it must be entered.

MAX_M_DROPOUT: Symbol or value used as the CML maximum M-contactor dropout time.

- Must be a real number (variable name or literal accepted) in seconds.
- This value represents the maximum time in seconds between a stop request (CML_RUN = FALSE) and the opening of the M-contactor. Two conditions will cause the M-contactor to be opened during a stop sequence: CML feedback indicates discontinuous conduction or the MAX_M_DROPOUT time expires. This parameter is only required for applications with exceptionally long motor electrical time constant values (Te) which may take more than .1 second (default value) for discontinuous conduction to be reached.
- Range = 0.1 through 1.0 second.
- Fixed at task initialization.
- Default = 0.1 second.

CML_COAST_STOP@: Symbol or value assigned as the CML coast-stop request input.

- Must be a boolean variable name (literal not accepted.)
- TRUE (ON) = coast-stop request.
- May be modified dynamically. Latched-in once every system clock tick while the drive is running (DC_CML_STATE@ = TRUE).
- Default = FALSE.

MCR_SELECT @: Symbol or value assigned as the motor contactor relay used on the drive digital I/O module when sequencing the DC drive on.

- Must be a boolean value (variable name or literal accepted).
- TRUE (ON) = RMCR (reverse MCR); FALSE (OFF) = FMCR (forward MCR).

- May be modified dynamically. Latched-in when sequencing the drive on (that is, rising edge of CML_RUN@ = TRUE).
- Default = FALSE (FMCR).

52.1.4 Test Mode Inputs

CML_TEST @: Symbol or value assigned as the CML test/normal switch input.

- Must be a boolean value (variable name or literal accepted).
- TRUE (ON) = CML test mode; FALSE (OFF) = CML normal mode.
- In test mode the CML will run open loop using the TEST_ALPHA input as the firing angle.
- May be modified dynamically. Latched-in once every system clock tick while the drive is in standby (DC_CML_STATE@ = FALSE).
- Default = FALSE (normal).

TEST_BRPOL @: Symbol or value assgined as the bridge polarity while in TEST mode.

- Must be a boolean value (variable name or literal accepted).
- TRUE (ON) = bridge #2 (reverse); FALSE (OFF) = bridge #1 (forward).
- May be modified dynamically. Latched-in once very system clock tick while the drive is in standby (DC_CML_STATE@ = FALSE).
- Default = FALSE [bridge #1 (forward)]

TEST_ALPHA%: Symbol or value assigned as the alpha firing angle while in TEST mode.

- Must be a single word integer (variable name or literal accepted), entered in degrees.
- Range = 0 to 180 degrees in steps of 1 degree.
- May be modified dynamically. Latched-in and processed once every system clock tick while the drive is running and in test mode (DC_CML_STATE@ = TRUE .AND. CML_TEST@ = TRUE).
- Default = 180 degrees.

52.1.5 Drive Faults Programming Inputs

IOC_THRESH%: Symbol or value assigned as the percentage of full-load motor current used as the instantaneous overcurrent threshold level.

$$\text{IOC\_THRESH\%} = \frac{\text{Instantaneous Overcurrent Threshold}}{\text{Full-Load Motor Current } (I_R)} * 100$$

- Must be a single word integer (variable name or literal accepted).
- Range = 100 through 400 in steps of 1.

- IOC_THRESH must be less than 1.8(LIM_BAR) or an IOC calculation error will occur at task initialization.
- Fixed at task initialization.
- No default exists for this input; it must be entered.

LIM_BAR%: Symbol or value assigned as the percentage of maximum current to full-load motor current.

$$\text{LIM\_BAR\%} = \frac{I_{max}}{I_{fl}} * 100$$

This input describes the current feedback scaling (10V/LIM_BAR).

- Must be a single word integer (variable name or literal accepted).
- Range = 100 through 400 in steps of 1.
- Fixed at task initialization.
- No default exists for this input; it must be entered.

WARNING

SPEED_FDBK IN SPEED REGULATOR CONFIGURATION MUST BE PROGRAMMED TO PROVIDE TACH LOSS PROTECTION. FAILURE TO OBSERVE THIS PRECAUTION COULD RESULT IN BODILY INJURY OR DAMAGE TO EQUIPMENT.

SPEED_FDBK%: Symbol assigned to the speed feedback point to be used when determining tach loss drive fault. Input must be scaled such that 4095 = 100% speed.

- Must be a single word integer variable name (literal not accepted).
- Latched-in during tach loss fault processing. FORCING of this input is not supported.
- No default exists for this input; if not entered, tach loss fault will not be detected or processed.

DIGITAL_TACH@: Symbol or value that indicates whether the digital tach speed feedback board is used with the drive. Used to configure the hardware overspeed circuitry.

- TRUE(ON) = digital tach; FALSE (OFF) = no digital tach.
- Must be a boolean value (variable name or literal accepted).
- Range = TRUE, FALSE.
- Fixed at task initialization.
- Default = TRUE.

TACH_LOSS_STOP@: Symbol or value assigned as the tach loss fault response. TRUE (ON) = initiate a coast-stop; FALSE (OFF) = indicate fault occurrence only.

- Must be a boolean value (variable name or literal accepted).
- Range = TRUE,FALSE.
- Fixed at task initialization.
- Default = TRUE (coast-stop on tach loss).

WARNING

IN SPEED REGULATOR CONFIGURATION, OSV_FDBK MUST BE PROGRAMMED TO PROVIDE OVERSPEED PROTECTION. FAILURE TO OBSERVE THIS PRECAUTION COULD RESULT IN BODILY INJURY OR DAMAGE TO EQUIPMENT.

OSV_FDBK%: Symbol assigned as the feedback point used when determining overspeed/overvoltage drive fault.

- Must be a single word integer variable name (literal not accepted).
- Latched-in during overspeed/overvoltage fault processing. FORCING of this input is not supported.
- No default exists for this input; if not entered, overspeed/overvoltage fault will not be detected or processed.

OSV_THRESH%: Symbol or value assigned as the overspeed/overvoltage threshold.

- Must be a single word integer (variable name or literal accepted).
- Range = 0 through 32767 in steps 1.
- Fixed at task initialization.
- Default = 0.

NONSTAND_PHROT@: Boolean value which selects the phase rotation direction of the drive.

- TRUE (ON) = nonstandard (forward rotation = A,B,C); FALSE (OFF) = standard (reverse rotation = C,B,A)
- Must be a boolean value (variable name or literal accepted).
- May be modified dynamically. Latched-in during incorrect phase rotation fault processing.
- Range = TRUE, FALSE.
- Default = FALSE (standard = reverse rotation = C,B,A).

AUTO_PHROT@: Boolean value which selects automatic phase rotation. Selecting auto phase rotation allows the drive to automatically compensate for a discrepancy between the programmed NONSTAND_PHROT value and the actual phase rotation determined by the DC drive I/O controller module hardware.

- TRUE (ON) = auto phase rotation; FALSE (OFF) = no auto phase rotation.
- Must be a boolean value (variable name or literal accepted).
- Range = TRUE,FALSE.
- Fixed at task initialization.
- Default = FALSE (no auto phase rotation).

FAULT_RESET @: Symbol assigned as the drive fault reset request signal.

- Must be a boolean variable symbol name (literal not accepted)
- May be modified dynamically. Latched-in once every clock tick while the drive is in standby (DC_CML_STATE = FALSE).

- No default exists for this input; it must be entered.

52.1.6 Drive Controller Module (57C406) Switch Outputs

These outputs are always updated once every system clock tick.

INSWITCH_POSA@: Symbol assigned to the signal representing the state of position A of the 3-position momentary test switch on the DC drive I/O controller module.

- TRUE (ON) = switch in position A; FALSE (OFF) = switch not in position A.

INSWITCH _POSB@: Symbol assigned to the signal representing the state of position B of the 3-position momentary test switch on the DC drive I/O controller module.

- TRUE (ON) = switch in position B; FALSE (OFF) = switch not in position B.

INFAULT_RESET @: Symbol assigned to the signal representing the state of the momentary 2-position fault reset switch on the DC drive I/O controller module.

- TRUE (ON) = switch closed; FALSE (OFF) = switch open.

52.2 Output Keyword Definitions

All outputs are optional and do not need to be programmed.

52.2.1 Loop Control Outputs

The following keywords define the names assigned to the available CML output values. They are always updated once every system clock tick.

DC_CML_STATE @: Symbol assigned to the signal representing the state of the Current Minor Loop.

- TRUE (ON) = run; FALSE (OFF) = standby.

SYNC_LOCKED @: Symbol assigned to the signal representing the state of the line sync Phase Locked Loop (PLL) function.

- TRUE (ON) = locked; FALSE (OFF) = not locked.

LINE_PERIOD%: Symbol assigned to the signal representing the value for line period, in µsec, computed by the PLL function.

- Nominal Range = 16129 (62 Hz) through 20833 (48 Hz) in steps of 1.

The following outputs are only updated while the CML is in the RUN mode. While in the STANDBY mode, they will retain the value they had during the last scan of the most recent RUN mode. They are set to zero (FALSE) when the task is initially turned on.

DC_TIME%: Symbol assigned to the value representing the measured time, in µsec, for which the feedback current = 0 since the last CML scan.

- Range = 0 through 3472 in steps of 1 µsec.

BRIDGE_POL @: Symbol assigned to the value representing the polarity of the power unit currently on.

- TRUE (ON) = bridge #2 (reverse); FALSE (OFF) = bridge#1 (forward).

REF_RATE_OUT%: Symbol assigned to the value representing the output of the CML reference rate limit function.

- Nominal Range = 4095 through −4095 in steps of 1.
- Maximum Range = 32767 through −32767 in steps of 1.

REF_LAG_OUT%: Symbol assigned to the value representing the output of the CML reference lag function.

- Nominal Range = 4095 through -4095 in steps of 1.
- Maximum Range = 32767 through −32767 in steps of 1.

FDBK_GAIN%: Symbol assigned to the value representing the CML feedback gain factor. This value is the gain in decimal times 1024. The feedback gain is chosen dynamically based on DC_TIME%.

- Range = 512 to 1280 in steps of 1:0.5 through 1.25 in steps of 1/1024.

I_FDBK%: Symbol assigned to the value representing the CML current feedback output signal.

- Nominal Range = 4095 through -4095 in steps of 1 (LIM_BAR through −LIM_BAR).
- Maximum Range = 10238 through -10238 in steps of 1.

INTEGRAL_GAIN%: Symbol assigned to the value representing the CML integral gain factor. This value is the gain in decimal times 1024. The integral gain is chosen dynamically based on DC_TIME%.

- Range = 0 through 32767 in steps of 1.0 through 31.999 in steps of 1/1024.

DELTA%: Symbol assigned to the value representing the delta value, in µsec, being used by the CML phase firing control (DELTA in degrees = 180 - ALPHA in degrees).

- Range = 0 through 10417 in steps of 1.

52.2.2 Drive Faults and Diagnostic Outputs

The following keywords define the names assigned to the available drive faults' output values. They are always updated once every system clock tick.

IOC_OUT @: Symbol assigned as the output state of the IOC drive fault.

- Must be a boolean variable name.

EXT_IET_OUT @: Symbol assigned as the output state of the external IET (instantaneous electronic trip) drive fault.

- Must be a boolean variable name.

SYNC_LOSS_OUT @: Symbol assigned as the output state of the line sync lost drive fault.

- Must be a boolean variable name.

TACH_LOSS_OUT @: Symbol assigned as the output state of the tach loss drive fault.

- Must be a boolean variable name.

OSV_OUT @: Symbol assigned as the output state of the over speed/overvoltage drive fault.

- Must be a boolean variable name.

PHROT_FLT_OUT @: Symbol assigned as the output state of the incorrect phase rotation drive fault.

- Must be a boolean variable name.

BAD_SCR_OUT @: Symbol assigned as the output state of the shorted SCR diagnostic drive fault. Programming this parameter enables the shorted SCR diagnostic procedure. Refer to section 45.3.1 Shorted SCR Diagnostic in this manual.

- Must be a boolean variable name.

BAD_SCR_NUM %: Symbol assigned as the number of the suspected shorted SCR. Refer to section 45.3.1 Shorted SCR Diagnostic in this manual.

- Must be an integer variable name.

The following output is updated only while the CML is in the RUN mode. While in the STANDBY mode, it will retain the value it had during the last scan of the most recent RUN mode. It is set to zero (FALSE) when the task is initially turned on.

DIAG_DATA_%: Array assigned as the output of the CML diagnostic data collection procedure. Refer to section 45.3.2 Diagnostic Data Collection in this manual.

- Must be a two-dimensional array dimensioned as 38 elements by 2 elements (37,1).
- When specifying this parameter in the CALL statement only, the array name without any subscript specification must be entered; for example, DIAG_DATA = diag_data%.
- The data declaration statement (LOCAL or COMMON) defines it as an array dimensioned (37,1); for example, LOCAL diag_data% (37,1).

52.3 Power Module Diagnostic Enhancements

The power module diagnostic enhancements provide the detection of a shorted SCR as well as a blown fuse (missing phase) and an open SCR gate lead. The DC_DRIVE_CML block provides the detection of a shorted SCR and also collects specific data and stores it into an array in a circular queue manner. This data can then be processed by a BASIC task running at a slower rate and lower priority.

52.3.1 Shorted SCR Diagnostic

Two output parameters support this portion of the diagnostic:

- BAD_SCR_OUT, which is the boolean flag indicating that a fault has been detected.
- BAD_SCR_NUM, which is an integer value indicating the number of the suspected shorted SCR.

The shorted SCR diagnostic is enabled by programming the parameter BAD_SCR_OUT. The parameter BAD_SCR_NUM is optional. If BAD_SCR_OUT is not programmed, the shorted SCR diagnostic will not be executed.

The diagnostic requires that individual gate leads be fired with the results evaluated each CML scan. It must be performed by the DC_DRIVE_CML block.

The diagnostic procedure is performed whenever a RUN request is made. This occurs on the rising edge of CML_RUN with all run permissive signals present. SYNC_LOCKED must be TRUE. Run permissive into the digital I/O must be TRUE and no drive faults can be active (TRUE).

The diagnostic consists of firing individual gates on the forward (#1) bridge for up to 9 line cycles. The current feedback as a result of each firing will be evaluated to determine if some sort of short circuit exists. The firing angle will begin at zero microseconds (0 degrees delta) and increase each line cycle up to a maximum of 800 microseconds or until a failure is concluded.

The Dierikon diagnostic option, applicable only to the Dierikon S6R module, fires SCRs in both the motoring and the regenerative bridge for up to 18 cycles of the line.

If the diagnostic concludes that no problem exists, DC_CML_STATE will be set TRUE. If this diagnostic is enabled, a delay of approximately 180 milliseconds between the RUN request and the assertion of DC_CML_STATE will be incurred.

A shorted SCR diagnostic failure is considered a drive fault and is treated as such. If a failure is detected, the following will occur:

- a boolean output from the CML (BAD_SCR_OUT@) will be set indicating the failure,
- an integer output from the CML (BAD_SCR_NUM%) will be set equal to the number of the suspected, and SCR, and
- fault code "87" will be displayed on the processor module LEDs, which will also be logged in the task's error log.

With an SCR configuration, the BAD_SCR_NUM parameter will indicate one of the two possible SCRs suspected of failing:

BAD_SCR_NUM	Suspected Shorted SCRs
1	1 or 14
2	2 or 15
3	3 or 16
4	4 or 11
5	5 or 12
6	6 or 13

A shorted SCR fault is a latched drive fault requiring a drive fault reset command to clear the fault condition. As with all drive faults, a start attempt (run request) will be ignored if any fault condition exists.

52.3.2 Diagnostic Data Collection

The DIAG_DATA output parameter supports the diagnostic data collection portion of the power module diagnostics. This parameter defines the storage array in which real time data is accumulated in a circular queue. (See Table 2.) It is an optional parameter and is unique to the DC_DRIVE_CML block in that it is a two-dimensional array. The array provides the interface between the current minor loop and a background task that processes this information to diagnose a missing line phase or an open SCR gate lead.

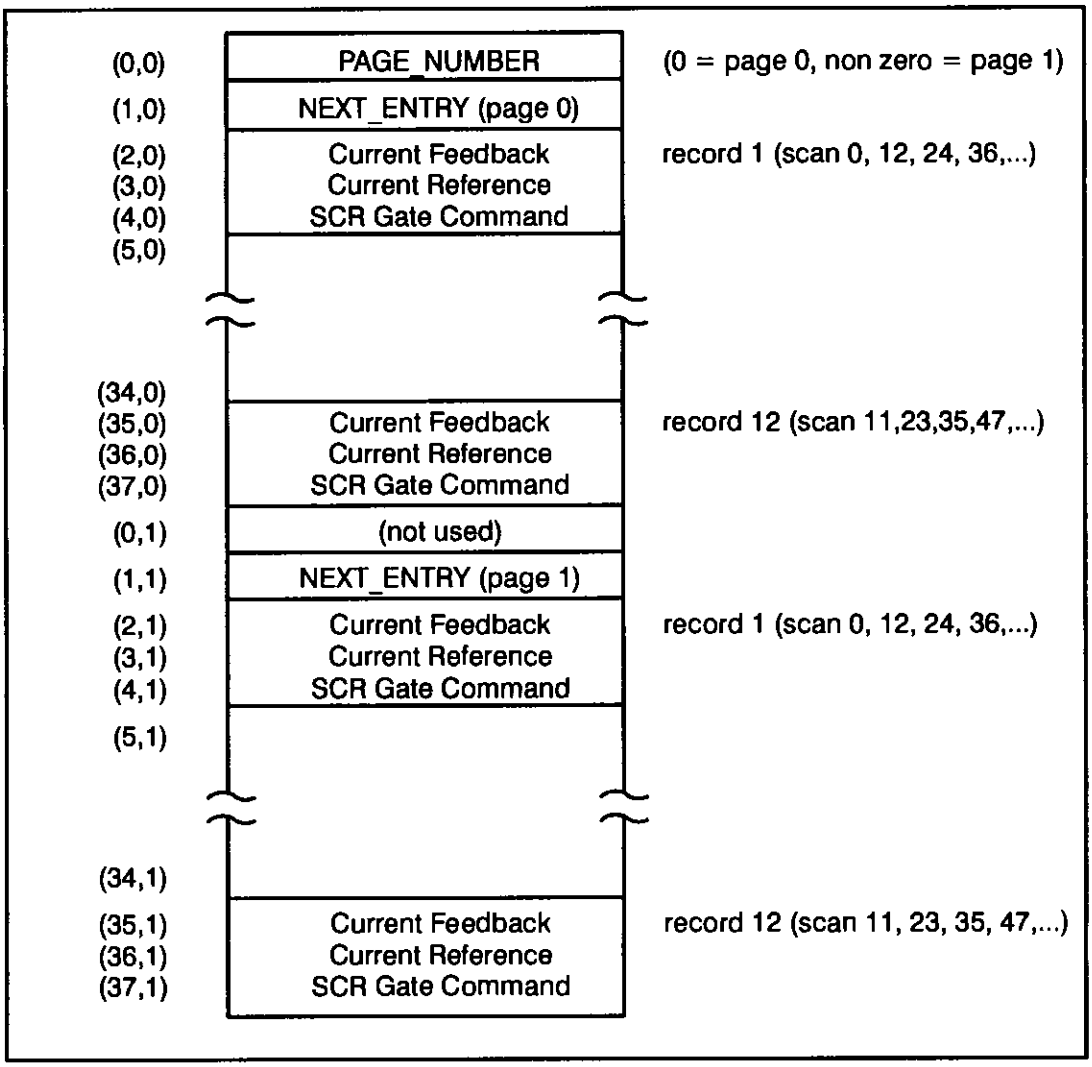
Consistent with the parameter definition, only the array name for the DIAG_DATA parameter and not the dimensions is entered (DIAG_-DATA = diag_data%). The array is dimensioned in the variable declaration statement:

COMMON diag_data%(37,1).

The dimensions must be 37 to 1 providing 38 by 2 elements.

While the drive is in the normal RUN mode, a data record is added to the DIAG_DATA array each CML scan. it is not written to while the drive is in STANDBY mode or TEST mode, or during a coast-stop sequence (Fault Stop). The CML scan period is nominally 2.778 milliseconds with a 60 Hz AC line frequency.

Table 2 - Organization of DIAG_DATA Array



The information provided in the array is defined as follows:

diag_date%(0,0) = PAGE_NUMBER

- This value defines which data 'page' is to be actively updated by the CML task.
- If diag_data%(0,0) = 0, the CML will update page 0 of the 2-dimensional array.
- If diag_data%(0,0) < > 0, the CML will update page 1 of the 2-dimensional array.

diag_data%(1,x) = NEXT_ENTRY

- This value is the number to the next data record to be written to the page by the CML. It will have a value between 1 and 12 inclusive.
- Before the PAGE_NUMBER is modified by the background task such that page x is used by the CML, NEXT_ENTRY must be set equal to zero (0).
- If NEXT_ENTRY is ever set to a value which is out of range, it will be clamped to 0 by the CML.

- NEXT_ENTRY, after swapping page x out, will equal the number of the next record which would have been written to on the next CML scan.

A data record contains 3 fields:

1. The first field is the latest current feedback signal. This is equal to I_FDBK currently provided as a CML output. This is the feedback signal at the loop summing junction. This field is subtracted from the second field which produces the CML error signal.
2. The second field is the latest current reference signal. This is equal to REF_LAG_OUT, currently provided as a CML output. This is the reference signal at the loop summing junction.
3. The third field is the gate command word indicating the SCR gates that are to be pulsed to produce the current feedback signal of the next scan.

52.4 Tach Loss Delta Threshold Adjustment

The tach loss fault function bases its decision on the logic:

If armature voltage > 40% and SPEED_FDBK < 5%, then tach loss fault is concluded.

Since armature voltage feedback is not always available, the average of 16 CML scans of the armature firing angle 'DELTA' is used to approximate armature voltage. The DELTA value is approximately equal to 40% armature voltage. It is calculated to be 109 degrees (alpha = 71 degrees). The value of SPEED_FDBK representing 5% speed is equal to 205. This assumes 1PN = 4095.

If these threshold levels are not adequate, tach loss faults may occur. Problems are most likely to occur in two situations:

1) A large Te motor having a large IR disp.
2) A severely notched or distorted AC line which yields less output voltage than expected for a given firing angle.

To provide for operation in these situations, the tach loss delta threshold may be increased.

52.4.1 Making Tach Loss Adjustments

A register on the drive I/O controller module (M/N 57C406) holds the tach loss delta threshold value specified in degrees. The register is set to its default value during initialization of the DC_DRIVE_CML block. If a change from the default value is required by the application, the statement to write to this register should be added to the initialization section of a task which is of lower priority than the DC_DRIVE_CML block task. (In most systems the CML task is the highest priority task.) Note that each time the CML task is restarted this register will be reinitialized by the DC_DRIVE_CML block.

WARNING

DO NOT ALTER THE FIRING ANGLE DEFAULT VALUE WITHOUT VERIFYING THAT THE DRIVE PARAMETERS REQUIRE SUCH MODIFICATIONS AS DEFINED IN THE SAFETY INSTRUCTIONS BELOW. FAILURE TO OBSERVE THESE PRECAUTIONS COULD RESULT IN BODILY INJURY OR DAMAGE TO EQUIPMENT.

The value for the tach loss delta threshold must be selected by using equations (1) and (2). Equation (1) gives the upper limit for the value and equation (2) determines if it is safe to make such a change.

1) DELTA THRESHOLD < = 180-ARCOS[(0.75*VDC) / (1.35*VLL)]

   DELTA THRESHOLD = Register 4136 on drive I/O controller (57C406). The value must be between 0 and 127, inclusive.

   VDC = Maximum armature voltage, (base speed armature voltage)

   VLL = Nominal AC line voltage feeding the drive.

   ARCOS = Arc cosine function, (inverse cosine)

2) JBAR > = 0.2 seconds [1]

   JBAR = (WR*WR*SB)/(308*T)

   WR^2 = Minimum rotational inertia (empty core) expressed in lb. $ft.^2$

   SB = Base speed of motor expressed in RPM. For the field weakened operation case, it is appropriate to use the maximum speed desired for the particular application. This will give conservative results since 100% torque is not usually available above base speed.

   T = Maximum torque expressed in ft. lbs. Maximum torque occurs at maximum armature current (LIMBAR).

   1 = If The time to reach the maximum safe speed for the application is less than 0.2 seconds, then no change in DELTA_THRESH is allowed.

Drive I/O Controller Board Register Number	Default Value	Description
4136	109	Tach loss delta threshold (degrees)

This register value is limited from 0 to 127 degrees inclusive by the tach loss function before its use. Note, however, that the limited value will not be overwritten into register 4136, i.e. if register 4136 is set outside of the 0 to 127 degree limit, the register value will not reflect the value used by the tach loss function.

Changes to the tach loss delta threshold (register 4136) can be made at any time and will be recognized by the tach loss function. Dynamic modification of this register, however, is of little use, except at system startup, and is strongly discouraged. Refer to section 45.7, the Drive I/O Controller Write Registers.

52.5 Line Synchronization Filter Adjustments

The purpose of the line sync phase lock loop (PLL) function, incorporated into the DC_DRIVE_CML block, is to stabilize the AC line synchronization point used by the drive's phase firing process. The PLL is required to compensate for drive-induced 'notching' and other noise disturbances in the AC line. The default adjustments to the PLL are such that its output moves very slowly due to perturbations on its input. For a stable AC line generator the default adjustments provide optimal line synchronization stability and tracking, requiring no PLL adjustments by the application software.

For applications where the AC power to the drive is provided by a local plant power generator (e.g. diesel alternator) or any power source which cannot maintain a 'suitable fixed' output frequency, adjustments to the PLL may be necessary. Problems will most likely become apparent when a step change in load is experienced by the power generator. When the step change in load is applied, the generator's output frequency will deviate from the desired output frequency until its regulator can correct for the error. During these transient periods, the PLL's output may not be able to change fast enough to follow the AC line. A gradual increase in phase shift occurs every cycle until the line returns to its original frequency or a fault trip limit is reached. To allow normal operation under these conditions, the PLL's filters for frequency and phase must have their limits changed to allow faster tracking of changes in the AC line's frequency.

52.5.1 PLL Filter Adjustments

Two PLL filter adjustments are available: one specifies the maximum change made in-phase and the other specifies the maximum change made in period, both in units of microseconds per line cycle.

Two registers on the drive I/O controller module (M/N 57C406) have been designated for this purpse. They are set to their default values during initialization of the DC_DRIVE_CML block. If changes from the default values are required by the application, the statements to write to these registers should be added in the initialization section of a task which is lower
priority than the DC_DRIVE_CML block task. (In most systems the CML task is the highest priority task.) Note that each time the CML task is restarted these registers will be reinitialized by the DC_DRIVE_CML block.

Drive I/O Controller Board Register number	Default Value	Description
4137	4	PLL maximum phase change (µsec)
4138	2	PLL maximum period change (µsec)

Neither value should be set less than or equal to zero or line sync loss faults will result. The value of the phase change limit (register 4137) must be twice the value of the period change limit (register 4138).

The greater these values become, the faster the PLL will compensate for changes in the AC line. This includes phase and frequency changes as well as noise disturbances. Therefore, as these values are increased, disturbances to the AC line may be reflected in phase firing control and ultimately the CML itself. Maintaining tight current control and operation without sync loss faults is the tradeoff which must be made when making PLL adjustments.

As a comparative guideline, increasing the maximum phase change limit (register 4137) to a value of 80 and the maximum period change limit (register 4138) to a value of 40 will provide a significant increase in PLL response. 80 microseconds is nearly 2 degrees relative to a 60 Hz line. Applications requiring values greater than these are considered extreme cases.

The largest phase change limit that will cause changes to the PLL function is 2000. Values greater than 2000 will have no more effect than a value of 2000 since the maximum phase error permitted by the PLL is fixed at 2000 microseconds.

52.6 Fast Bridge Change

The deadtime for systems without the fast bridge change feature can range from 3 milliseconds to 70 milliseconds. Fast bridge change reduces the deadtime to approximately 2 milliseconds. Deadtime is directly related to the motor's CEMF when the bridge change request is made and the amplitude of the current reference signal. The bridge change deadtime is defined as the time from reaching zero current on one bridge to producing current on the other bridge.

There are three requirements in order to provide the fast bridge change feature. First, the hardware must provide an armature voltage feedback signal to the drive analog I/O board (M/N 57C405). Second, the application software must enable the fast bridge change logic. Third, the motor's inductance must be small enough to allow safe fast bridge changes without crossfires. See Table 4.

Table 3 - Maximum Armature Inductance for Fast Bridge Change

RATED I	230VAC 240VDC	460VAC 500VDC	575VAC 600VDC	660VAC ← Line Voltage 700VDC ← Max. Drive Voltage
1500.0	0.8431	1.5644	2.1079	2.3466
1400.0	0.9034	1.6761	2.2584	2.5142
1300.0	0.9729	1.8051	2.4321	2.7076
1200.0	1.0539	1.9555	2.6348	2.9332
1100.0	1.1497	2.1332	2.8743	3.1999
1000.0	1.2647	2.3466	3.1618	3.5199
950.0	1.3313	2.4701	3.3282	3.7051
900.0	1.4052	2.6073	3.5131	3.9110
850.0	1.4879	2.7607	3.7197	4.1410
800.0	1.5809	2.9332	3.9522	4.3998
750.0	1.6863	3.1288	4.2157	4.6932
700.0	1.8067	3.3522	4.5168	5.0284
650.0	1.9457	3.6101	4.8643	5.4152
600.0	2.1079	3.9110	5.2696	5.8664
550.0	2.2995	4.2665	5.7487	6.3998
500.0	2.5294	4.6932	6.3236	7.0397
475.0	2.6626	4.9402	6.6564	7.4102
450.0	2.8105	5.2146	7.0262	7.8219
425.0	2.9758	5.5214	7.4395	8.2820
400.0	3.1618	5.8664	7.9045	8.7997
375.0	3.3726	6.2575	8.4314	9.3863
350.0	3.6135	6.7045	9.0337	10.0568
325.0	3.8914	7.2202	9.7286	10.8303
300.0	4.2157	7.8219	10.5393	11.7329
275.0	4.5990	8.5330	11.4974	12.7995
250.0	5.0589	9.3863	12.6471	14.0794
225.0	5.6209	10.4292	14.0524	15.6438
200.0	6.3236	11.7329	15.8089	17.5993
190.0	6.6564	12.3504	16.6410	18.5256
180.0	7.0262	13.0365	17.5654	19.5548
170.0	7.4395	13.8034	18.5987	20.7051
160.0	7.9045	14.6661	19.7611	21.9991
150.0	8.4314	15.6438	21.0785	23.4657
140.0	9.0337	16.7613	22.5842	25.1419
130.0	9.7286	18.0506	24.3214	27.0759
120.0	10.5393	19.5548	26.3482	29.3322
110.0	11.4974	21.3325	28.7435	31.9988
100.0	12.6471	23.4658	31.6178	35.1986
90.0	14.0524	26.0731	35.1309	39.1096
80.0	15.8089	29.3322	39.5223	43.9983
70.0	18.0673	33.5225	45.1683	50.2838
60.0	21.0785	39.1096	52.6964	58.6644
50.0	25.2943	46.9315	63.2356	70.3972
40.0	31.6178	58.6644	79.0445	87.9966
30.0	42.1571	78.2192	105.3927	117.3288
20.0	63.2356	117.3288	158.0891	175.9931
10.0	126.4713	234.6575	316.1781	351.9863

RATED I = THE MOTOR'S RATED CURRENT FOR 100% TORQUE

ALL VALUES ARE IN AMPERES AND MILLIHENRIES.

52.6.1 Hardware Requirements

To provide the fast bridge change feature, hardware must provide and isolate the scale armature voltage. Armature voltage must be scaled according to the following table and the result fed into the drive analog I/O module (M/N 57C405) through connector P4 from pin 5 (–) to pin 12 (+).

ARMATURE VOLTAGE TRANSFORMER SCALING

Transformer rating (Vrms)	Nominal voltage (Vrms)	Scaling to 57C405	
		Vin (Vrms) @Vo=5V	Gain
240	230	250	02
397	380	413.54	0121
480	460	500	01
575	550	598.96 (~600)	00835
690	660	718.75	00696

52.6.2 Software Requirements

The fast bridge change software enable switch is located on the drive I/O controller module (M/N 57C406) at bit 0 of register 4146. This switch is initialized to the default value of 'off' (0) during initialization of the DC_DRIVE_CML_ block. To enable the fast bridge change feature, an application task of lower priority than the CML task must set this bit (register 4146, bit 0) to 'on' (1).

The state of register 4146, bit 0 is latched by the CML task while the drive is in standby. Therefore, enabling and disabling the fast bridge change feature can only be accomplished while the drive is in standby (DC_DRIVE_STATE = false). Note that each time the CML task is restarted the fast bridge change enable bit will be reinitialized (turned off) by the DC_DRIVE_CML block. If the fast bridge change software is enabled in an application which does not have the required hardware, an IOC fault will occur.

52.7 Drive I/O Controller Write Registers

The drive I/O controller module (B/M 57C406) contains the following registers which an application task can write to.

Register No.	Default Value	Description
4136	109	Tach loss delta threshold (degrees)
4137	4	PLL maximum phase change (µsec)
4138	2	PLL maximum period change (µsec)
4146 bit 0	0	Fast bridge change enable
4146 bit 1	0	SCR diagnostics for Dierikon S6 and Industrial Controls' S6 & S6Rs; set to 1 for Dierikon S6R DC drives
4146 bit 2-15	n/a	(undefined)

Accesses to the drive I/O controller module (M/N 57C406) delay execution of the microprocessor on that module. Therefore, accesses (reads and writes) made by the application software to the M/N 57C406 module must be kept to a minimum and only made when absolutely necessary. Towards this end, it is strongly recommended that any accesses to the above registers be made during the initialization section of an application task(s). Note that the application task must be a BASIC task and not a control block task. The reason for this is that control tasks will latch the state of all common variables at the start of each scan and store them locally, regardless of whether any run-time references to those variables are made. Creating a separate BASIC task which does nothing but initialize the above write registers to the values required by the application will minimize references to the M/N 57C406 module.

53.0 EXECUTION TIME ESTIMATES

Execution time can be affected by remark statements within a task. There are two forms of the remark statement: REM and !. Remark statements using "REM" are stripped off before the task is downloaded and, therefore, do not affect execution time. However, remark statements using "!" do require execution time when the task runs on the Processor. For tasks that will run on 6010/6011 Processors, use a value of 40 microseconds as the time for each "!" remark. For tasks that will run on 7010 Processors, use a value of 9.88 microseconds as the time for each "!" remark. For tasks that will run on UDC modules, use 2.37 microseconds as the time for each "!" remark. The amount of text that follows the "!" does not affect the execution time; no text requires the same time as a full line of text. For computing CPU usage for the task, only those remark "!" statements that occur after the SCAN_LOOP block need be considered. Those that occur before the SCAN_LOOP block are executed only once and, therefore, need not be considered.

53.1 AutoMax Processor and AutoMax PC3000 Control Block Tasks

The execution times for all control block statements programmed in a particular AutoMax task can be summed with the SCAN_LOOP/END time to estimate the total task execution time. This value divided by the scan time (TICKS * tick rate in seconds) gives the estimated CPU usage for the control block task. Refer to Table 4 for AutoMax Processor block execution times. Refer to Table 6 for AutoMax PC3000 block execution times. Note that the times listed are valid for Version 3.4 and later of the AutoMax Programming Executive software.

53.2 UDC Tasks

The execution times for all Control Block statements programmed into a particular UDC Control Block tasks can be summed with the SCAN_LOOP/END time to estimate execution time for that UDC Control Block task. If two Control Block tasks will be running on a particular UDC module, the estimated execution times of both tasks must be added together. The total execution time divided by the tick rate (.5 milliseconds) will equal the minimum value you should enter for the TICKS parameter in the SCAN_LOOP block. **Note that you must enter the same value for the TICKS parameter in the SCAN_LOOP block for both tasks.** The maximum value you can enter for the TICKS parameter is 20 (10 milliseconds). Refer to Table 5. Note that the times listed are valid for Version 3.4A of the AutoMax Programming Executive software.

Table 4 - Maximum Execution Time Summary - AutoMax Tasks

Block Name	7010 Processor Maximum Time (µsec)	6010/6011 Processor Maximum Time (µsec)
ABSOLUTE_VALUE	9.16	41
ALARM	11.16 + n(2.48)	44 + n(11)
AMPLIFIER	13.16 + k(3.12)	62 + k(13.8)
AND	10.80 + j(1.88)	41 + j(8.3)
BIT_SELECT	26.04 + n(1. 20)	118 + n(6.5)
COMPARE	11.24 + n(1. 16)	45 + n(5.5)
COUNTER	15.64	61
DIFFERENCE	10.80	44
DIFF_LAG	33.04	154
END	13.82 + t (2.0) + i (1.6) + b(2.2) + d(2 .5)	60 + t (3.2) + i(2.8) + b(4 .3) + d(4 .9)
FUNCTION	23.56	93
INTEGRATE	39.14	193
INVERTER	10.84	48
LAG	26.52	118
LATCH	15.04	68
LEAD_LAG	68.60	284
LIMIT	18.20	69
MOVE	10.40 + p(2.32)	39 + p(11)
MULTIPLY_DIVIDE	13.76	71
NOTCH	85.16	393
OR	10.24 + j(1. 88)	41 + j(8.3)
PACK_BITS	24.28 + j(1. 44)	110 + j(5)
PID	124.00	515
PROP_INT	40.32	193
PULSE_MULT	18.32	86
RAMP	27.72	122
READ_BITS	24.32 + n(1.12)	106 + n(7.3)
READ_WORDS	25.76 + n(1. 28)	111 + n(5.6)
RUNNING_AVERAGE	28.64	125
SAMPLED_AVERAGE	29.00	125
S_CURVE	47.00	282
SCALE	22.48	98
SCAN_LOOP	80.00 + t(1.6) +i(2.5) + b(3 .8) + d (4.1)	326 + t(2.4) + i(4.4) + b (6.8) + d (7.2)
SEARCH	19.80 + m(0.68)	74 + m(3)
SELECT	11.36 + j (2.52)	44 + j (12.2)
SHIFT_BITS	18.28 + p(2.04)	92 + p(19.7)
SHIFT_WORDS	17.80 + p(l.72)	89 + p(13.3)
SUMMER	11.28	44
SWITCH	11.52	47
TRANSITION	11.36	48
UNPACK_BITS	23.40 + n(1. 44)	106 + n(7.3)
WRITE_BITS	25.56 + j (1.36)	103 + j (6)
WRITE_WORDS	25.52 + J(0.36)	110 + j(1.9)
!<REM>	9.88	40

b = number of common boolean variables referenced by the task
d = number of common double integer variables referenced by the task
i = number of common integer variables referenced by the task
j = total number of inputs programmed
k = total number of input pairs programmed
m = number of elements searched
n = total number of outputs programmed
p = total number of input/output pairs programmed
t = total number of common integer, boolean, and double integer variables referenced by the task

Table 5 - Maximum Execution Time Summary - UDC Tasks

Block Name	UDC Module Maximum Time (µsec)
ABSOLUTE_VALUE	8.30
ALARM	10.10 + n(1.80)
AMPLIFIER	15.30 + k(1.80)
AND	10.60 + j(1.45)
COMPARE	7.50 + n(2.25)
COUNTER	16.90
DIFFERENCE	7.97
DIFF_LAG	41.50
END	130.00 + i(2.40) + b(2.50) + d(3.30) +z
FUNCTION	20.30
INTEGRATE	58.60
INVERTER	9.40
LAG	41.00
LAGN	39.25 + o(4.00)
LEADN	39.25 + o(4.00)
LATCH	14.85
LEAD_LAG	39.70
LIMIT	15.05
MOVE	10.40 + p(1.80)
MULTIPLY_DIVIDE	13.80
NOTCHN	64.25 + o(20.00)
OR	8.90 + j(1. 45)
PACK_BITS	16.16 + n(1.25)
PROP_INT	70.85
PULSE_MULT	18.96
RAMP	24.96
RUNNING_AVERAGE	26.94
SAMPLED_AVERAGE	25.94
S_CURVE	35.01
SCALE	18.30
SCAN_LOOP	350.00 + i(2.20) +b(2.00) + d(3.30) + x
SELECT	12.50 + j(1.50)
SUMMER	10.00
SWITCH	12.55
TACHLOSS_OVERSPEED	22.00
THERMAL_OVERLOAD	46.10
TRANSITION	11.90
UNPACK_BITS	14.40 + n(1.90)
I<REM>	2.37

b = number of common boolean variables referenced by the task
d = number of common double integer variables referenced by the task
i = number of common integer variables referenced by the task
j = total number of inputs programmed
k = total number of input pairs programmed
n = total number of outputs programmed
o = "order" of filter
p = total number of input/output pairs programmed
x = avg. feedback message overhead per task; add 312 µsec when only 1 task executes.
z = avg. setpoint message overhead per task; add 100 µsec when only 1 task executes

Table 5 - Maximum Execution Time Summary - AutoMax PC3000 Tasks

Block Name	PC3000 Maximum Time (µsec)
ABSOLUTE_VALUE	6.44
ALARM	7.8 + n(1.76)
AMPLIFIER	9.2 + k(2.20)
AND	7.56 + j(1.32)
BIT SELECT	18.24 + n(0.84)
COMPARE	7.88 + n(0.80)
COUNTER	10.96
DIFFERENCE	7.56
DIFF LAG	23.12
END	9.68 + t(1.40) + i(1.12) + b(1.56) + d(1.76)
FUNCTION	16.48
INTEGRATE	27.40
INVERTER	7.6.0
LAG	18.56
LATCH	10.52
LEAD LAG	48.04
LIMIT	12.76
MOVE	7.28 + P(1.64)
MULTIPLY_DIVIDE	9.64
NOTCH	59.60
OR	7.16 + j(1.32)
PACK_BITS	17.00 + j(1.00)
PID	86.80
PROP_INT	28.24
PULSE_MULT	12.84
RAMP	19.40
READ_BITS	17.04 + n(0.80)
READ_WORDS	18.04 + n(0.92)
RUNNING_AVERAGE	20.04
SAMPLED_AVERAGE	20.32
S_CURVE	32.92
SCALE	15.72
SCAN_LOOP	56.00 + t(1.76) +b(2.68) + d(2.88)
SEARCH	13.88 + m(0.48)
SELECT	7.96 + j(1.76)
SHIFT_BITS	12.80 + p(1.44)
SHIFT_WORDS	12.48 + p(1.20)
SUMMER	7.88
SWITCH	8.08
TRANSITION	7.96
UNPACK_BITS	16.40 + n(1.00)
WRITE_BITS	17.88 + j(0.96)
WRITE_WORDS	17.88 + j(0.28)
!<REM>	6.92

b = number of common boolean variables referenced by the task
d = number of common double integer variables referenced by the task
i = number of common integer variables referenced by the task
j = total number of inputs programmed
k = total number of input pairs programmed
m = number of elements searched
n = total number of outputs programmed
p = total number of input/output pairs programmed
t = total number of common integer, boolean, and double integer variables referenced by the task

Forward To: Reliance Electric – RGA
Technical Publications
25001 Tungsten Road
Cleveland, OH 44117

Technical Writing Internal Use:

DIF # _____

V★S DRIVES & INDUSTRIAL CONTROLS DOCUMENTATION IMPROVEMENT FORM

Document Number: _____

Page Number(s): _____

Comments: (Please give chapters, page numbers or specific paragraphs that the change will affect. Include markups from the document or attach additional pages if necessary.)

What will this improvement suggestion provide? _____

Originator: _____ City: _____ State: _____ ZIP: _____

Company: _____ Phone: (_____) _____

Address: _____ Date: _____

Technical Writing Internal Use:

Follow-Up Action:

Writer: _____ Date: _____

RE 1857LC2 Printed in U.S.A.

Thank you for your comments . . .

Rockwell *Automation*

Reliance Electric

Reliance Electric / 24703 Euclid Avenue / Cleveland, Ohio 44117 / 216-266-7000

Rockwell Automation

Reliance Electric

Printed in U.S.A. J-3676-5 November, 1996