IB-11B022

MACROPRO II™ FOR WINDOWS™ DECEMBER 1998

MACROPRO II™

FOR WINDOWS™

INSTRUCTION BOOK

INDUSTRIAL INDEXING SYSTEMS, Inc.

Revision - B
Approved By:

TABLE OF CONTENTS

MACROPRO II™

FOR WINDOWS™

Section 1 - Introduction

INDUSTRIAL INDEXING SYSTEMS, INC.

SECTION 1 - INTRODUCTION

Table of Contents

This page is intentionally left blank.

Introduction

This document is one in a series of technical documents supporting Industrial Indexing Systems DeltaMax Servo Controller. This particular document provides information about the MacroPro II Development System for Windows™, which serves as a tool to assist the user in the development of motion control programs.

Overview

The MacroPro II Development System is intended to be used with the DeltaMax programmable motion controllers.

The MacroPro II Development System runs under the Microsoft Windows™ environment, providing users a very powerful, flexible and intuitive software approach to program generation and testing.

This manual was written with the assumption that you are somewhat familiar with Personal Computers, MS-DOS and the Microsoft Windows environment.

Highlights

1. The Integrated Development Environment (IDE) provides access to the major MacroPro II features.

2. The MacroPro II Help System allows:

   a. Access to on-line documentation whenever needed.

   b. Search for topic on keyword, history, next/previous.

   c. The ability to cut and paste examples or documentation into your program.

3. The MacroPro II Editor offers:

   a. A multi-window, multi-file editing environment.

   b. Access to the MacroPro II Compiler while editing your program.

   c. Access to the MacroPro II Analyzer.

   d. Automatic syntax verification of your program lines.

   e. Unique color combinations for instructions, comments and errors.

   f. Standard Windows based file management features.

   g. On-line documentation.

4. The MacroPro II Compiler provides:

   a. Program compiler errors that are clear and concise.

   b. Compiler errors are easily identified and can be located within your program by "double clicking" on the error description.

   c. The ability to abort compiling and the ability to resume compiling.

5. The MacroPro II Analyzer provides a real time test and analysis facility, including:

   a. Controller and application program status.

   b. View and change DeltaMax System User Parameters.

   c. Program trace features.

   d. The ability to select data variables and flags from a sorted list.
   e. The ability to monitor data variables only, flags only or mixed data and flags.

f. Simultaneous reading and writing of data variables and flags.

g. Axis controller status flags can be monitored in real time.

h. A controller information screen provides information on the hardware/firmware in use.

i. The ability to view your source and symbol files.

Manual Conventions

Throughout this manual, the following typeface conventions are used:

1. Program instruction names appear in **bold** print.

2. Optional instruction labels appear in *italics*.

Example 1

The following program line defines a timer (called MONITOR) with a time of 100.

```
wait_1_sec      set_tmr      MONITOR,100
```

The label *wait_1_sec* is an optional label. Although it may be needed because the user intends for the program to branch to that label during program execution, it is not required for the instruction **set_tmr** to be valid.

Example 2

The first line of the following group of program instructions, is an example of an instruction which requires an instruction label.

```
speed_table     begin_data
                data          100,150,200,600,1200
                end_data
```

The label **speed_table** is a required label. This label identifies a place in the program data area, where an array of data can be retrieved. Without the label, we have no way of accessing the data.

Getting Started

The first step to developing and testing your programs is to install the MacroPro II Development System on your computer. While this is a relatively simple task, it is important to understand the recommended and minimum hardware configuration of your Personal Computer. Once your software has been successfully installed, take a few minutes to read the section which follows regarding the program development cycle. This should give you some insight into the major components of the MacroPro II Development System and how the pieces fit together.

System Requirements

System

A 386 class machine is required minimally. A 75MHz Pentium class machine with at least 16MB of RAM is recommended. Performance will also vary depending on the video adapter, driver and video mode you have selected.

Memory

MacroPro II requires 4MB of memory to operate. Additional memory may be required depending on the size of the programs being developed.

Disk Space

When fully installed, MacroPro II uses approximately 5MB of disk space. Additional disk space may be required depending on your application(s).

Windows

You must be running Microsoft Windows Version 3.1 or greater in enhanced mode or Microsoft Windows 95.

Installing MacroPro II

Like many Windows based applications, a program called SETUP.EXE resides on the MacroPro II Installation Disk and will guide you through the installation process. Basically, you will do the following steps:

Microsoft Windows Version 3.1 or greater

1. Begin Windows by typing **win** at the MS-DOS prompt and pressing the Enter key. You should be running Windows version 3.1 or greater.

2. Insert the disk labeled Disk 1 in your floppy disk drive.

3. Choose the Run command from the File menu in Windows Program Manager. Type **a:setup** and press the Enter key (or **b:setup** if using the drive b:).

4. Follow the instructions on the screen and provide the appropriate responses to the installation options as needed.

Microsoft Windows 95

1. Begin Windows and make sure the user can see the Windows Desktop. Select the START button and click RUN on the START menu.

2. Insert the disk labeled Disk 1 in your floppy disk drive.

3. Type **a:setup** and press the Enter key (or **b:setup** if using the drive b:).

4. Follow the instructions on the screen and provide the appropriate responses to the installation options as needed.

Once the installation process is complete, you should be able to run the MacroPro II Development System by selecting the corresponding icon or running from the Program Manager.

Release Information

MacroPro II Release	Firmware Number
v1.X	SFO-3901
v2.X	SFO-3902
v3.X	SFO-3903

Application Development

The MacroPro II Development System provides you with the necessary tools for creating, compiling and testing your motion control programs. A description of the major components of the MacroPro II Development System follows.

Selecting a function using the IDE

The MacroPro II IDE (Integrated Development Environment) provides easy access to the MacroPro II Editor, Analyzer and Help System. Selections are made by pressing the desired button on the IDE screen. Selecting the Exit button will close down the MacroPro II Development System.

Creating a Motion Control Program

Application programs are created and/or changed using the MacroPro II Editor. The Editor has a wealth of features that will help streamline the editing of your application programs. The actual text that you enter is often referred to as the **source** file.

Compiling Your Program

The program **source** file will need to be compiled before it can be run on your DeltaMax Controller. The process of compiling must be initiated by you, although the actual process of compiling is transparent to you.

The Compiler is initiated from the Editor. By pressing the Compile button from the Toolbox, the compile process will begin. The **source** file currently being edited, will be saved. Then, the Compiler checks for syntax errors in your **source** file and builds other files that will be required to run your program.
Typically, the Compiler will build a **symbol** file and an **error** file. These files are built regardless of whether an error was or was not encountered. An optional **listing** file might also be created.

A **trace** file and **executable** file will also be created **if no compile errors were discovered**. The Compiler will always indicate whether a compile was successful, unsuccessful or was aborted by the programmer.

Since the **executable** file will only be created if there were no Compiler errors, it follows that only compiled programs without errors can be run on the DeltaMax Controller.

Testing Your Program

Application programs are tested using the MacroPro II Analyzer. The Analyzer is used to transmit, start, stop and autostart your application program at the DeltaMax. The Analyzer also provides the tools to watch program variables, trace program execution and monitor the status of the DeltaMax Controller and related axes, I/O and other events.

If the Analyzer is selected from the Editor, the currently selected file will be used **provided that file has been successfully compiled and is error free**. Other application programs to be run can be selected while in the Analyzer assuming that they are error free.

While in the Analyzer, you will have easy access to all program **source** and **symbol** file information which will prove useful as your program is being tested.

Running Your Program

Once your program has been successfully tested, you are ready to setup your DeltaMax Controller so that the program will run automatically on system power-up. There is one way to do this.

Through the Analyzer select the "Actions" topic, then select "Run w/ AutoStart". Your application program will run automatically each time power to the DeltaMax Controller is enabled.

MACROPRO II™

FOR WINDOWS™

Section 2 - Programming Language Concepts

INDUSTRIAL INDEXING SYSTEMS, INC.

SECTION 2 - PROGRAMMING LANGUAGE CONCEPTS

Table of Contents

This page is intentionally left blank.

Programming Language Concepts

A program is an organized group of statements that can be executed by a DeltaMax Controller. Application programs reside in a section of non-volatile memory in the controller.

Memory allocation is as follows:

Program Area:	64,000 bytes
Integer Data Area + Float Data Area:	16,000 bytes
Integer Constants Area + Float Constants Area:	16,000 bytes
Extended Memory - Motor Axis:	27,000 bytes
Extended Memory - Pseudo Axis:	27,000 bytes

Program Statement Format

Program *statements* are, in many ways, similar to the statements of the BASIC computer language. Each statement may consist of up to 4 parts; a *label*, an *instruction*, a list of *parameters*, and an optional *comment*. This format is illustrated in Figure 2.1.

1. *label* - A label can be from one (1) to twelve (12) characters in length, and may consist of both upper and lower case letters, numbers and the underscore character. Labels must begin in column 1 of the program line. Labels are optional. If no label is used, at least one blank must be at the beginning of the line.

2. *instruction* - This part of the line consists of any valid MacroPro II Programming Language command. Instructions must be typed in lower case letters. At least one blank must precede the instruction and at least one blank must follow the instruction.

3. *parameters* - This part of the program line consists of the information, if any, processed by the instruction. It must be separated from the instruction by one or more spaces.

4. *comment* - This field can contain any explanatory information about the program line. It must be separated from the preceding field by at least two spaces. Program lines which contain only comments must have an exclamation point (!) in column 1.

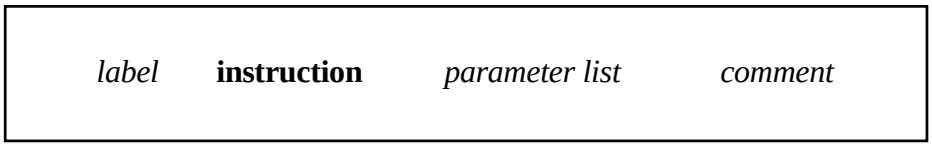


Figure 2.1 - Program Instruction Format

The label portion of the statement is not always required, depending on the type of instruction used. It is required for instructions which define variables, text strings, arrays, or equated values.

There are instructions which do not require a list of parameters. However, there are usually one or more parameters for an instruction. In the example in Figure 2.2, two parameters are required.

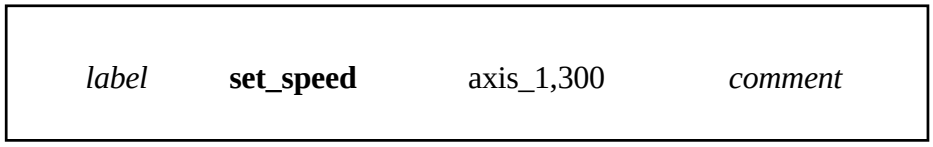


Figure 2.2 - Instruction With Two Parameters

Comments

Comments provide a means for a programmer to describe the functions being performed by a particular instruction or group of instructions. It is important to note that comments do not use any of the storage within the program memory space. The liberal use of comments is highly recommended. There are two ways to implement comments in a program:

1. An entire line of text can be dedicated as a comment line by placing an exclamation point (!) in the first character position of the line.

2. Comments can be placed at the end of an instruction line by leaving at least two spaces between the last parameter in the instruction and the beginning of the comment.

Comments of both formats are included in the sample program shown in Figure 2.3.

Blank Lines

Blank lines may be entered in the program to make it easier to read. For example, a subroutine might be separated from the main body of the program by one or more blank lines. As with comments, blank lines do not use any storage within the program.

Label Lines

Program lines containing only a label are allowed. It is often handy to place a line containing only a label just ahead of an instruction it references. In this way, it is easier to insert new instructions in the program if necessary during the testing process. Lines containing only a label use no storage within the program.

```
LABEL           OPERATION       PARAMETERS      COMMENT

                declare         ON
POS_1           equ             81920           two turns CW
POS_2           equ             -81920          two turns CCW
LIFT            equ             1               use axis 1
LIFT_DOWN       equ             93              down flag
LIFT_BUSY       equ             94              busy flag
NO_FAULT        equ             0               I/O zero

! do initialization functions

                turn_on         NO_FAULT        show no fault

                drive_on        LIFT            enable drive
                set_speed       LIFT,50         500 rpm speed
                set_ac_dc       LIFT,20         rev/sec/sec

! set a local zero at current position

                set_local       LIFT

! begin main program loop

restart
                position        LIFT,POS_1      go 2 turns CW
loop1           if_stat_on      LIFT_DOWN,fault check error
                if_stat_on      LIFT_BUSY,loop1 wait for done

                position        LIFT,POS_2      go 2 turns CCW
loop2           if_stat_on      LIFT_DOWN,fault check error
                if_stat_on      LIFT_BUSY,loop2 wait for done

                goto            restart         do it all again

fault           turn_off        NO_FAULT        signal error
                sys_return
```

Figure 2.3 - Sample Application Program

Program Compiler

The MacroPro II Compiler converts DeltaMax application programs from human readable instructions **(source files)** to numeric codes **(executable files)** which can be interpreted and acted upon by the DeltaMax Controller.

In addition, the Compiler produces information which is used by the MacroPro II Editor and MacroPro II Analyzer to simplify the programming and testing processes. The user can specify that an optional listing file be created when desired.

The Compiler is invoked by the user from the MacroPro II Editor, by pressing the Compile button.

Configuring the Compiler

The user may request that a **listing** file be created during the compile process. This is typically a rather large file containing a considerable amount of detail regarding the compiled file. This information is of little importance to most users. It is, however, available to those users with a need to know.

A **listing** file will be created, if the "List file=ON" string is read from the "[Compiler]" section of the file MPRO2.INI. This file will be found in your WINDOWS directory.

A **listing** file will not be created if the "List file=OFF" string is read from the "[Compiler]" section of the file MPRO2.INI.

By default, a **listing** file will not be created.

The MPRO2.INI File

The Compiler will attempt to read the MPRO2.INI file from the WINDOWS directory at the start of the compile process. This file should minimally contain lines similar to those listed below.

The following lines indicate the directory containing the file MPRO2.OPC.

[Environment]
Exec dir=c:\directory

The following lines indicate the path and name of the last file used.

[Last File]
Last file=c:\progdir\name.prg

The following lines indicate whether a listing file should be produced at compile time.

[Compiler]
List file=ON

Compile Time Problems

Initiating the Compiler is a very simple task. However, there are a number of conditions which might cause the Compiler to abort. The Compiler will attempt to read the MPRO2.INI file from the WINDOWS directory at the start of the compile process. The process will be aborted, with an error message, for any of the following reasons:

1. The MPRO2.INI file was not found in the WINDOWS directory.

2. The "[Environment]" section was not found.

3. The "Exec dir=" string was not found within the "[Environment]" section.

4. The MPRO2.OPC file was not found in the directory listed in the "[Environment]" section.

5. The path and filename passed to the Compiler is illegal or does not exist.

Should you encounter any of these problems, verify that your MPRO2.INI file is located in the proper directory and has the correct syntax as described above.

Aborting the Compiler

Once the compile process has begun, you may abort that process if necessary. A dialog box will be displayed, indicating the current pass of the Compiler and the percent complete for that pass (the Compiler is a four pass process, where passes 2 and 4 do most of the work). An ABORT button will also be shown.

Press this button to abort the compile process at any time. The Compiler will display a message box, requesting that you confirm whether to abort or not. Press the YES button to confirm the abort or NO to resume compiling. The Compiler will pick up where it left off, if you decide to resume.

Compiling is Complete

When the compile process is complete, a message box will be displayed indicating that the compile was successful or not successful.

If successful, the Program Size (in bytes) and the Data Size (in bytes) will be displayed.

If unsuccessful, a message will be displayed indicating the number of errors detected.

In either case, you must press the OK button in the message box in order to continue.

Compiler Output

The MacroPro II Compiler will generate the following files:

1. The **listing file** is an optional file. It will be created if specified in the MPRO2.INI file. The listing file has the program file extension **.lst**. It will be created even if program errors were detected.

   The **listing file** consists of:

   a. An EQUATE Table - This table lists all symbols defined in **equ** statements. Its contents are symbol name, decimal symbol value, and hexadecimal symbol value.

   b. A LABEL Table - This table contains all program statement labels and their equivalent addresses.

   c. A CONSTANTS Table - This consists of a list of all constants used. This table consists only of addresses and values.

   d. A DATA Table - This table contains all the data variables used in the program. Each entry in this table consists of the variable name, followed by its address in both decimal and hexadecimal formats.

   e. A PROGRAM Listing - This portion consists of a listing of each program line together with the DeltaMax numerical equivalent of the program line. Any errors detected in a program line will be listed at the end of the listing file.

2. An **error file** is always created and always includes some header information. When errors are detected, they are written to this file. The information regarding each error includes; program source file, line number and error description. The **error file** has the program file extension **.err**. It is used by the MacroPro II Editor to assist in locating and correcting errors.

3. A **symbol file** is always created. This file contains all symbol, label, and data definitions. The **symbol file** has the program file extension **.sym**. It is used by the Analyzer to assist in the testing and analysis process.

4. The **executable file** is only created when no compiler errors are detected. This file contains the compiled program in DeltaMax machine instruction format. The **executable file** has the program file extension **.mcx**. The Analyzer transmits this file to the DeltaMax Controller where it will be executed.

5. The **trace file** is only created when no compiler errors are detected. This file contains the information needed by the Analyzer to translate trace information into source program lines. This file has the program file extension **.dbg**. The **trace file** is only used by the Analyzer.

Compiler Directives

Compiler directives are a class of instructions which simplify such operations as defining constants, data arrays and cams, or which cause the MacroPro II Compiler to function in a particular way. These instructions have no direct effect on the DeltaMax Controller. Their purpose is to simplify the programmer's task.

A list of compiler directive instructions and their formats is shown in Figure 2.4.

Instruction	**Format**		
begin_data	label	**begin_data**	
begin_cam	label	**begin_cam**	
cam		**cam**	value,value,etc.
data		**data**	value,value,etc.
declare		**declare**	mode
dim	label	**dim**	size
dim	label	**dim**	controller#,size
end_cam		**end_cam**	
end_data		**end_data**	
equ	label	**equ**	value
float	label	**float**	
float_dim	label	**float_dim**	size
integer	label	**integer**	
text	label	**text**	"ASCII string"

Figure 2.4 - Compiler Directives

MACROPRO II™

FOR WINDOWS™

Section 3 - Control and Motion Concepts

INDUSTRIAL INDEXING SYSTEMS, INC.

SECTION 3 - CONTROL AND MOTION CONCEPTS

Table of Contents

This page is intentionally left blank.

Flags

A flag is a bit in memory representing the current state of a timer, axis status, I/O or other condition. The DeltaMax has reserved a storage area in memory for 256 flags. A flag can be either on (1) or off (0).

There are a number of program instructions dealing with flags. In fact, taken as a group, flag instructions may be one of the largest groups of program instructions.

There are flag instructions to read inputs, to turn outputs on and off, to start and stop timers, to read motor activity and fault conditions and to read, set and clear user program switches.

Figure 3.1 lists the flags used for the DeltaMax Controller.

Flag Categories

The following are the major categories of flags:

DeltaMax Internal Status Flags

FLAG #	FUNCTION	CATEGORY
0 - 15 16 - 23 24 - 31 32 - 71	INPUTS ONLY OUTPUTS OR HARDWARE PLS SOFTWARE PLS FLAGS RESERVED FLAGS	INPUT/ OUTPUT
72 - 79	PROGRAMMABLE TIMERS	TIMER
80 - 95 96 - 111 112 - 119 120 - 127 128 - 207	CONTROLLER #1 STATUS FLAGS PSEUDO AXIS STATUS FLAGS ANALOG STATUS FLAGS EXTENDED STATUS FLAGS NOT USED	CONTROLLER STATUS
208 - 255	GENERAL PURPOSE USER FLAGS	USER

Figure 3.1 - DeltaMax Internal Status Flags

Timers

DeltaMax Controllers provide 8 user programmable timers. These timers have a resolution of 10 milliseconds per "tick".

Timers are used by first setting the timer to the desired number of counts or ticks. This enables the flag associated with the timer. It will remain enabled until the specified time has passed.

Controller Flags

The following are categories of various controller flags:

DeltaMax Controller Status Flags

#1	DESCRIPTION
80	NOT USED
81	LOCK PENDING
82	FOLLOWING ERROR
83	MASTER/TEST MODE
84	CAM/PW PROFILE SIZE EXCEEDED
85	COMMAND INVALID IN THIS STATE
86	NOT USED
87	OVERDRAW FAILED
88	JOGGING
89	INDEXING
90	CALCULATION IN PROGRESS
91	HARDWARE INTERRUPT ARMED
92	FORCED DECEL IN PROGRESS
93	DOWN
94	BUSY
95	MASTER/SLAVE LOCK

Figure 3.2 - DeltaMax Controller Status Flags

DeltaMax Pseudo Axis Controller Status Flags

#2	DESCRIPTION
96	NOT USED
97	NOT USED
98	NOT USED
99	NOT USED
100	CAM/PW PROFILE SIZE EXCEEDED
101	COMMAND INVALID IN THIS STATE
102	NOT USED
103	NOT USED
104	JOGGING
105	INDEXING
106	CALCULATION IN PROGRESS
107	NOT USED
108	FORCED DECEL IN PROGRESS
109	NOT USED
110	BUSY
111	NOT USED

Figure 3.3 - DeltaMax Pseudo Axis Controller Status Flags

Flag Descriptions

LOCK PENDING

This flag is set during lock modes 3, 8 or 9 between the time that the lock instruction is issued and the time that the master angle crosses the specified angle. The angle is specified in the **set_trig_pw** command for lock mode 3 or **set_trig_cam** for lock mode 8 or 9 and can be crossed from either direction.

FOLLOWING ERROR

A following error occurs when the difference between the actual transducer angle and the expected transducer angle is outside the allowable range. The angles are compared every 100 ms. The allowable error is +/- 17 degrees if the motor shaft is motionless and +/- 180 degrees if the motor shaft is moving.

MASTER/TEST MODE

This flag will be set if the axis has been put into the Master/Test mode. All motor fault conditions will have been reset. This is the default, power on state for the DeltaMax Controller. The servo amplifier will have been disabled and no checks for servo following errors will be made.

CAM SIZE EXCEEDED/PIECEWISE PROFILE ARRAY FULL

This flag will be set if the number of cam array data elements exceeds 28K bytes, or if the number of piecewise profile segments exceeds 99.

COMMAND INVALID IN THIS STATE

This flag will be set whenever the axis is requested to execute a command that cannot be executed at that moment. This typically occurs when program steps are out of logical order. An example would be the situation where the axis is currently jogging and is then commanded to **index** without first completing a **f_decel** instruction and waiting for the motor to stop.

OVERDRAW FAIL

This flag will be set when in the position profile the overdraw speed was not reached.

JOGGING

This flag will be set when a **jog_cw**, **jog_ccw**, **track_spd**, **l_track_spd**, **vel_cw** or **vel_ccw** command is executed. This flag will be cleared after completing a **f_decel** operation.

INDEXING

This flag will be set when a **index** or **position** command is executed. This flag will be cleared after completing the selected move.

CALCULATING PIECEWISE PROFILE

This flag will be set while a Piecewise Profile is being calculated. It will be cleared otherwise.

CALCULATING PLS DATA

This flag will be set while Programmable Limit Switch data is being calculated.

HARDWARE INTERRUPT ARMED

The axis controller has received a hardware interrupt instruction and is monitoring its respective hardware interrupt input line.

FORCED DECEL IN PROGRESS

This flag will be set when an **f_decel** or **unlock** command is executed and the axis is still in motion.

DOWN

This flag will be set if a FOLLOWING ERROR is encountered (FOLLOWING ERROR is described above).

BUSY

This flag will be set when the axis is busy performing a motion related operation, for example **jog_cw**. If an axis card experiences a **DOWN** condition while the **BUSY** flag is set, **BUSY** will remain set.

MASTER/SLAVE LOCK

This flag will be set when a **lock** command is executed. It will be cleared when a **f_decel**, **unlock**, or **drive_off** command is executed.

BUSY PROCESSING

This flag is set when the axis task handler is executing a lengthy command to prevent the program instruction task handler from initiating another program instruction for the specified axis. The following program instructions cause this flag to be set: **calc_cam_sum, calc_unit_cam, drive_on, drive_off, over_draw, delta_comp** and **prep_profile.**

Flag Instruction Summary

Instruction	Format		
blk_io_in	*label*	**blk_io_in**	input flag#,variable
blk_io_out	*label*	**blk_io_out**	output flag#,variable
clr_flag	*label*	**clr_flag**	user flag#
if_flag_off	*label*	**if_flag_off**	user_flag#,address_label
if_flag_on	*label*	**if_flag_on**	user_flag#,address_label
if_io_off	*label*	**if_io_off**	I/O flag#,address_label
if_io_on	*label*	**if_io_on**	I/O flag#,address_label
if_stat_off	*label*	**if_stat_off**	status_flag#,address_label
if_stat_on	*label*	**if_stat_on**	status_flag#,address_label
if_tmr_off	*label*	**if_tmr_off**	timer_flag#,address_label
if_tmr_on	*label*	**if_tmr_on**	timer_flag#,address_label
set_flag	*label*	**set_flag**	user_flag
set_tmr	*label*	**set_tmr**	timer_flag#,ticks
set_ms_tmr	*label*	**set_ms_tmr**	timer_flag#,ticks
turn_off	*label*	**turn_off**	I/O flag#
turn_on	*label*	**turn_on**	I/O flag#

Figure 3.4 - Flag Instructions

Arithmetic Instructions

Several types of arithmetic functions are available, including:

1. 32 bit integer arithmetic
2. 64 bit floating point (double) arithmetic
3. Array manipulation
4. Byte operations
5. Bit set and bit clear operations
6. Absolute, negate and square root functions

Each of these items will be explained in more detail below.

Arithmetic Overview

There are a variety of arithmetic-based operations and support functions which are supported in MacroPro:

- Integer arithmetic.
- Floating point (double) arithmetic.
- Integer variables (treated as 32-bit / 4 byte locations).
- Floating point variables (treated as 64-bit / 8 byte locations).
- Arithmetic operations
  - Addition +
  - Subtraction -
  - Multiplication *
  - Division /
  - Shift logical left <<
  - Shift logical right >>
  - Logical AND &
  - Logical OR |
  - Rotate left {{
  - Rotate right }}

  Integers Only
- Arithmetic comparators
  - Assignment =
  - Greater than >
  - Greater than or equal to >=
  - Less than <
  - Less than or equal to <=
  - Not equal to <>

- Arithmetic functions
  - Square root **let** x = SQR(y)
  - Absolute value **let** z = ABS(value)
  - Negate **let** t = NEG(r)

- Arrays/memory allocation
  - Simple memory allocation

```
array_name    dim        100
.
.
label         let        array_name[x] = variable
```

  - Axis Controller memory allocation using 28K volatile storage

```
array_name    dim        axis_controller_#, #_of_4-byte_words
.
.
label         let        array_name[x] = variable
              or
label         let_byte   array_name[x] = variable
```

- Arithmetic assignment using simple equations

```
label   let     a = b
label   let     a = b + 5
label   let     a = b / 1789.123
```

- Arithmetic compare/jump using simple equations

```
label   if      x < 10, under_ten
```

- Arithmetic assignment of single byte variables

```
label   let_byte    array_name[0]=pos
label   let_byte    pos = array_name[0]
```

- Bit manipulation instructions

```
label   set_bit     bit_number, variable
label   clr_bit     bit_number, variable
```

- Bit compare/jump instructions

```
label   if_bit_set  bit_number, variable, addr_label
label   if_bit_clr  bit_number, variable, addr_label
```

Integer and Floating Point Arithmetic

Program variables will occupy 32 bits (4 bytes) of storage when defined as integers and 64 bits (8 bytes) when defined as floats. Arithmetic statements operate on these 4 or 8 byte variables. The format for arithmetic instructions is similar to that of the **LET** statement used in BASIC:

label **let** **result = variable1 + variable2**

where **result** is the result of the operation, **variable1** and **variable2** are the variables to be operated on. The symbol for addition "+" is used for purposes of illustration. Note that multiple operations in a single **let** statement are *not allowed*.

For example,

label **let** **a = b + c**

is acceptable, but

label **let** **a = b + c / d**

is not.

Calculations requiring multiple operations must be performed using multiple **let** statements. Spaces between variable names and operators should not be used.

Mixing data types is allowed. For example, **result**, **variable1** and **variable2** above can be defined as floats, integers or both. The operating system will automatically convert the variable types as it deems necessary. When assigning a floating point calculated result to an integer variable, the fractional portion will be truncated. No rounding will take place.

Array Manipulation

A data array is a group of 32 or 64 bit variables which can be referenced by the same variable name. Data arrays are usually defined using a **dim** or **float_dim** instruction or by use of the **begin_data**, **data**, and **end_data** instructions. Data array subscripts are zero based. The **INSTRUCTION SUMMARY** provides additional information on these instructions.

For example, to set aside ten 32 bit storage locations to contain a table of positions, the instruction

position **dim** **10**

could be used. Within the program, the first position in the table would be referred to as **position[0]**. The value within brackets is known as the array subscript. Note that for a table of ten values, the array subscript ranges from 0 to 9. Array subscripts can be a constant, an expression, or an integer variable. They cannot be floating point variables.

Special forms of the **let** instruction serve to store information in and retrieve information from data arrays. In our position table example, the instruction

label **let** **p=position[3]**

would retrieve the fourth position from the array and store that value in the variable **p**. To store a value in a data array, the instruction

label **let** **position[3] = x**

would be used.

To use a value from an array in arithmetic instructions, it is necessary to retrieve the value into a non-array variable, perform the arithmetic operation, and then store the result back in the array.

Byte Operations

Special instructions for the manipulation of byte oriented data such as electronic cams or strings of characters. This special instruction, the **let_byte**, is similar to the special case of the **let** statement for handling data arrays. For example, to access the fourth character of the text string **f_name**, the instruction

label **let_byte** **a = f_name[3]**

could be used. To store data into a byte oriented data array, the instruction format

label **let_byte** **f_name[3] = a**

would be used.

The **let_byte** instruction does not support any arithmetic operations. To perform arithmetic on byte oriented data, it is necessary to retrieve the byte into a conventional variable, perform the arithmetic, and store the result back into the byte array. The **let_byte** instruction treats byte information as unsigned.

Bit Oriented Operations

A set of instructions are available for setting (turning on) and clearing (turning off) individual bits within a 32 bit data value. These instructions can be useful when setting up variables for use with the **set_map** and **set_mcf** instructions. A complementary set of bit testing instructions allow program branching to take place depending on the state of a single bit in a 32 bit data value.

These bit oriented instructions cannot be used with variables declared as floating point.

Arithmetic Functions

A select group of arithmetic functions are included for commonly required calculations. These functions are listed in Figure 3.5. The format of these functions is:

label **let** **result = function(x)**

where **function** represents the function name.

Function	Description
abs(x)	Returns the absolute value of x.
sqr(x)	Returns the square root of x. NOTE: If x is negative or zero, zero is returned.
neg(x)	Returns the two's complement of x.

Figure 3.5 - Arithmetic Functions

Arithmetic Instruction Summary

Instruction	Format		
clr_bit	*label*	**clr_bit**	bit#,variable
dim	*label*	**dim**	100
dim	*label*	**dim**	controller#,100
float_dim	*label*	**float_dim**	45
float	*label*	**float**	
if	*label*	**if**	op1 *compare* op2, jump_address
integer	*label*	**integer**	
let_byte	*label*	**let_byte**	destination=source
let	*label*	**let**	variable=op1 *operation op1*
set_bit	*label*	**set_bit**	bit#,variable

Figure 3.6 - Arithmetic Instructions

Program Flow Instructions

Normally, program instructions execute in an orderly, sequential fashion. Sometimes, it is desirable to alter this sequential order based on the status of an I/O module, the result of a calculation, the detection of a motor fault, or some other condition. Program flow instructions provide this capability.

There are three general classes of program flow instructions - branching, subroutine control, and the select statement. A special type of program flow instruction based on interrupts is described separately.

Branching Instructions

There are two types of branching instructions - unconditional and conditional. The **goto** and **restart_at** instructions are the only unconditional branch instructions other than subroutine control instructions, which are covered separately.

Conditional branching instructions are handled as follows:

1. Test the specified condition.

2. If the condition is true, transfer program control to the specified address.

3. If the condition is false, continue by executing the next sequential instruction.

Conditional instructions can test the status of DeltaMax flags, the result of an arithmetic comparison, the status of bits in a variable, or whether characters are present at a serial port.

Subroutine Control

When designing a program, it is often desirable to place a group of commonly used instructions into a unit called a subroutine. The subroutine can then be called from anywhere within the program. Subroutines must begin with a labeled statement. The label on this initial statement is often referred to as the subroutine name. Subroutines must end with a **return_sub** instruction.

The **gosub** instruction is then used to transfer program control to the subroutine. The subroutine instructions are then executed until the **return_sub** instruction is encountered. At that point, program control returns to the instruction immediately following the **gosub** instruction.

The Select Statement

Often, the process of controlling program flow is more complex than testing a flag and branching if the flag is set. For example, it may be necessary to choose among several alternatives based on a user entered menu selection. In cases like these, the **select** instruction group provides an effective means of controlling program flow.

Figure 3.7 illustrates how the **select** instruction might be used.

```
label     select          keynum

                          case            1
                                  .
                                  (statements to do when keynum = 1)
                                  .
                          exit_select

                          case            2
                                  .
                                  (statements to do when keynum = 2)
                                  .
                          exit_select

                          default
                                  .
                                  (statements to do when keynum does
                                  not equal a case option)
                                  .
                          exit_select

          end_select
```

Figure 3.7 - Select Statement Usage

A program may contain up to 32 **select** statement groups. Each group may contain up to 100 **case** instructions.

Program Branching Instructions

Instruction	Format		
case	*label*	**case**	number
default	*label*	**default**	
end_select	*label*	**end_select**	
exit_select	*label*	**exit_select**	
gosub	*label*	**gosub**	subroutine label
goto	*label*	**goto**	address_label
if	*label*	**if**	value1 op value2,address_label
if_bit_clr	*label*	**if_bit_clr**	bit#,value,address_label
if_bit_set	*label*	**if_bit_set**	bit#,value,address_label
if_char	*label*	**if_char**	port#,address_label
if_flag_off	*label*	**if_flag_off**	user_flag#,address_label
if_flag_on	*label*	**if_flag_on**	user_flag#,address_label
if_io_off	*label*	**if_io_off**	I/O flag#,address_label
if_io_on	*label*	**if_io_on**	I/O flag#,address_label
if_no_char	*label*	**if_no_char**	port#,address_label
if_stat_off	*label*	**if_stat_off**	status_flag#,address_label
if_stat_on	*label*	**if_stat_on**	status_flag#,address_label
if_tmr_off	*label*	**if_tmr_off**	timer_flag#,address_label
if_tmr_on	*label*	**if_tmr_on**	timer_flag#,address_label
restart_at	*label*	**restart_at**	address_label
return_sub	*label*	**return_sub**	
select	*label*	**select**	value

Figure 3.8 - Program Branching Instructions

Motion Instructions

Motion instructions are divided into five classes:

1. Pre-motion Preparation
2. Velocity Control
3. Incremental and Absolute Positioning
4. Piecewise Profiles
5. Master/Slave Operations

Motion Preparation Instructions

Position Data

DeltaMax Controllers maintain position data as a 32 bit number. The least significant 12 bits of this number represent the position of the feedback transducer. The most significant 20 bits are used as a signed turns counter. The range of valid positions is from -524,288 to +524,288 turns, assuming 4096 bits per motor revolution. Positive positions represent a displacement from zero in the clockwise direction.

Incremental distances are expressed in a manner similar to position data. For example, an incremental distance of two turns would be expressed as 8192 (2*4096). Positive distances represent clockwise motion.

Speed (Velocity) Data

DeltaMax Controllers process speed data in RPM. Speeds may range from 0 to 5800 RPM in whole number increments. In special cases, speed values can be scaled down by a factor of 256 to provide fractional speed control.

Acceleration Data

Acceleration (accel/decel) data is expressed in revolutions/second^2. Accelerations may range from 1 to 1600 revolutions/second^2 . In special cases, acceleration values can be scaled down by a factor of 256 to provide very slow acceleration rates.

Global And Local Zeroes

DeltaMax Controllers can maintain both a global and a local zero. Global zero usually refers to a fixed machine home position, and is often established by moving a load until it contacts a sensor. Local zero can be thought of as a floating home position. Local zero is often used when moving a known distance from the global zero, setting the local zero, and then performing a series of motions relative to the new zero position. The local zero may then be cleared.

Motion Preparation Instructions

Motion preparation instructions are used to condition a controller prior to executing any motion. They are used to turn on the drive (amplifier), to set speeds and acceleration rates, and to set and clear zero positions.

On power up, DeltaMax Controllers set their turn counters to zero. The low order 12 bits of its position are also set to zero.

Instruction	Format		
clr_local	*label*	**clr_local**	controller#
delta_comp	*label*	**delta_comp**	controller#,AJ2,AJ3,AJ4,AJ7,AJ8,AJ9
drive_off	*label*	**drive_off**	controller#
drive_on	*label*	**drive_on**	controller#
set_ac_dc	*label*	**set_ac_dc**	controller#,rate
set_gl_ccw	*label*	**set_gl_ccw**	controller#
set_gl_cw	*label*	**set_gl_cw**	controller#
set_local	*label*	**set_local**	controller#
set_speed	*label*	**set_speed**	controller#,speed

Figure 3.9 - Motion Preparation Instructions

Digital Compensation

The DeltaMax Controller provides a means of digital compensation at the drive, through the use of a **delta_comp** command. The programmer has access to the following drive parameters:

ADJUSTMENT PARAMETER	SYMBOL	SETTING RANGE	DEFAULT SETTING	DESCRIPTION
LOAD INERTIA RATIO	**AJ2**	0-00.0 Times	1.0 Times	Sets the baseline frequency response of the driver using the ratio of the load inertia/motor inertia for a rigidly coupled load. If the load is not rigidly coupled, the value entered may vary from the calculated value. If the value is set too high, the motor and driver may become unstable and oscillate.

HIGH FREQUENCY RESPONSE	**AJ3**	0.1-20.0	1.0	Sets the high frequency response of the driver. The higher the number the more responsive. If the value is set too high, the motor and driver may become unstable and oscillate. The value in AJ3 is unitless and works in concert with AJ2.
POSITION LOOP DC GAIN	**AJ4**	1-200 RAD/SEC	30 RAD/SEC	Sets the DC gain of the position control loop. A higher value in AJ4 results in stiffer, faster response. If the value is set too high, the motor and driver may become unstable & oscillate.
ZERO SPEED GAIN REDUCTION	**AJ7**	0-10000	0	Sets the amount of gain reduction at zero speed.
FEED FORWARD GAIN	**AJ8**	0-2.0 TIMES	1.0 TIMES	Sets the feed forward gain in the position loop. A value of 50% results in 0.0 following error. Less than 50% will produce a lag between the actual motor position and the commanded position and greater than 50% produces a lead. The lead or lag will be proportional to speed at non 50% settings.
NOTCH FILTER FREQUENCY	**AJ9**	100-20000 RAD/SEC	6000 RAD/SEC	Sets the notch frequency of a velocity loop anti-resonance filter. This filter can be used to cancel machine or servo resonance. Power must be turned OFF then ON for this parameter to take effect.

Table 3.1

The **delta_comp** values are based on 0 being the minimum value of the corresponding drive parameter and 100% being the maximum value.

The default values will be loaded and stored in the drive each time the controller is reset. When a new **delta_comp** instruction is issued the new values will be sent to the drive.

Velocity Control Instructions

Velocity control instructions are used in applications where control of motor speed is the objective. Velocity control instructions are listed in Figure 3.10.

DeltaMax Controllers provide several different types of velocity control instructions. Choice of instruction is usually determined by the requirements of the application. Each of the different types is described below.

1. Jog instructions cause the axis controller to run its motor at a constant velocity until commanded to stop. Direction of rotation is determined by the instruction -- **jog_cw** or **jog_ccw**. Velocity may not be changed while the motor shaft is moving.

2. Track speed instructions allow on-the-fly speed changes. Unlike jog instructions, direction of rotation is determined by the sign of the speed variable, with positive values indicating clockwise rotation. Two forms of the track speed instruction are provided. The **track_spd** instruction covers speeds in the range of -3600 to 3600 RPM with a resolution of 1 RPM. The **l_track_spd** instruction covers speeds from -128 to +127.99 RPM in increments of 1/256 RPM.

3. Velocity instructions provide speed control with very low acceleration rates. Like the jog instructions, direction of rotation is determined by the instruction syntax. When a controller receives a **vel_cw** or **vel_ccw** command, it divides its present acceleration rate by 256 before using it. Speed, but not direction, may be changed while in velocity mode by issuing another velocity instruction with the new speed.

4. To stop a motor shaft, the **f_decel** instruction is used. This instruction causes the motor to decelerate to zero speed at the currently programmed deceleration rate.

Instruction	**Format**		
f_decel	*label*	**f_decel**	controller#
jog_ccw	*label*	**jog_ccw**	controller#
jog_cw	*label*	**jog_cw**	controller#
l_track_spd	*label*	**l_track_spd**	controller#,speed
track_spd	*label*	**track_spd**	controller#,speed
vel_ccw	*label*	**vel_ccw**	controller#
vel_cw	*label*	**vel_cw**	controller#

Figure 3.10 - Velocity Control Instructions

Positioning Instructions

DeltaMax Controllers support both absolute and incremental positioning. Absolute motions are made relative to the zero or home position established by the controller. Incremental motions are made relative to the current motor position. An absolute move is referred to as a **position**. An incremental move is called an **index**.

Instruction	Format		
incr_com	*label*	**incr_com**	controller#,bits,interrupts
index	*label*	**index**	controller#,distance
mod_index	*label*	**mod_index**	controller#,speed,rate,distance,degree
mod_position	*label*	**mod_position**	controller#,speed,rate,abs_position,degree
position	*label*	**position**	controller#,abs_position
set_offset	*label*	**set_offset**	controller#,offset

Figure 3.11 - Positioning Instructions

Piecewise Profiles

Piecewise Profiles provide a simple means of defining complex motion profiles as a series of profile segments. Each segment consists of a target speed, accel/decel rate and distance.

DeltaMax Controllers perform Piecewise Profiles according to the following guidelines:

A. If the present motor speed is LESS than the speed specified by the next profile segment, then:

1. Accelerate to the new speed.
2. Continue at the new speed until the distance specified has been moved. NOTE - some of the distance is used during acceleration.

B. If the present motor speed is GREATER than the speed specified by the next profile segment, then:

1. Continue at the old speed until the specified distance less the distance needed to decelerate has been traveled.
2. Decelerate to the new speed.

Building Profile Data Tables

Profile data tables are organized as a series of profile segments. Each profile segment consists of a target speed (in RPM), an acceleration value (in revs/second^2) and a distance (in bits). The last profile segment in a profile must contain a zero speed. All profile segments in a given profile must contain direction values of the same sign.

Probably the most convenient method of creating Piecewise Profile tables would be to use the **data** statement. The following defines a three segment profile.

```
prodata     begin_data
            data        500,100,6.2*4096
            data        1000,200,8.6*4096
            data        0,150,15.2*4096
            end_data
```

The profile above can be described as follows:

1. Accelerate to 500 RPM at an acceleration rate of 100 revs/sec^2, continue until the motor has turned 6.2 revolutions (including acceleration distance).

2. Accelerate to 1000 RPM at an acceleration rate of 200 revs/sec^2. Continue at that speed until the motor has traveled 8.6 additional turns (including acceleration distance).

3. Continue at 1000 RPM as long as necessary, then decelerate to zero speed at a deceleration rate of 150 revs/sec^2. The distance traveled in this segment is 15.2 motor revolutions (including deceleration distance).

4. The total distance traveled in the profile is 30 revolutions.

The following program statements could be used to execute the segments discussed above.

Program Example

```
!
! ----------  FLAG DEFINITION ----------
!
CALCULATI       equ             90
NG
BUSY            equ             94

!
! ----------  SEGMENT DATA ----------
!
pw_data         begin_data
                data            500,100,6.2*4096
                data            1000,200,8.6*4096
                data            0,150,15.2*4096
                end_data
                .
                .
!
! ----------  CALCULATE THE PROFILE ----------
!
                prep_profile    1,pw_data               send profile
ck_calc         if_stat_on      CALCULATING,ck_calc     wait for calcs
                get_pstat       1,status
                if              status<>0,calc_error    error routine
                .
!
! ----------  BEGIN RUNNING THE PROFILE ----------
!
                exec_profile    1                       do the profile
wt_busy         if_stat_on      BUSY,wt_busy            wait til done
                .
                (instructions performed when execution is ok)
                .
!
! ----------  ERROR HANDLER  ----------
!
calc_error      let_byte        segment=status[3]       get segment in error
```

```
let_byte        error=status[2]          get error code
.
(process errors here)
.
.
```

Piecewise Profiles may also be calculated in the program and stored in an array in the appropriate format. This format is always:

speed, accel, distance	**Segment #1**
speed, accel, distance	**Segment #2**
.	.
.	.
.	.
speed, accel, distance	**Segment #x**

The maximum number of profile segments allowed in a Piecewise Profile is 96. Note that the last segment in a Piecewise Profile must have a zero speed.

In the DeltaMax Controllers, Piecewise Profile data shares the same memory area as cam data. Piecewise profile data is always placed in the axis extended memory area starting at location zero.

Piecewise Profiles And Master Slave

It is possible to send a Piecewise Profile to the axis extended memory and to instruct the controller to execute the profile whenever an external device reaches a specified angular position. Refer to the section on **LOCK METHOD 3** in **MASTER SLAVE CONCEPTS** for a discussion of this technique.

Instruction	Format		
exec_profile	*label*	**exec_profile**	controller#
get_pstat	*label*	**get_pstat**	controller#,status
prep_profile	*label*	**prep_profile**	controller#,array_label
set_trig_pw	*label*	**set_trig_pw**	controller#,master_angle

Figure 3.12 - Piecewise Profile Instructions

Reading Controller Position

DeltaMax Controllers provide instructions for reading the current transducer position of the axis controller, as well as the position of the "pseudo axis." It is possible to read two types of position data from an axis controller:

1. Actual Position - The **get_pos** instruction returns the actual position of the corresponding transducer.

2. Commanded Position - The **get_com** instruction returns the commanded position for the specified controller. Commanded position is usually slightly different than actual position. The difference is referred to as following error.

NOTE: The "**pseudo axis**" commanded position is always equal to actual position.

Instruction	Format		
get_com	*label*	**get_com**	controller#,variable
get_fol_err	*label*	**get_fol_err**	controller#,variable
get_pos	*label*	**get_pos**	controller#,variable

Figure 3.13 - Instructions to Read Controller Position

Master Slave Concepts

The DeltaMax Controller provides a mode of operation whereby the motion of one axis, called a master axis, can be used to control the motion of a slave axis and/or Programmable Limit Switches.

A Slave Axis can use the Master Axis position to determine its own position and speed according to various methods described in subsequent sections of this chapter.

In the DeltaMax, the Master Axis transmits its current position transducer reading onto the designated data bus every 1 millisecond. A Slave Axis or Programmable Limit Switch listening on that bus will retrieve this position and use it to determine its own position. This high speed data rate allows for very accurate tracking and smooth operation.

There are currently three modes of operation for the slave axis. In simple lock, or "electronic gearbox" mode, a simple ratio is applied to the master position/speed data and the results used to directly determine the proper slave axis position and speed. In electronic cam mode, the slave axis moves through a pre-programmed path (the electronic cam) dependent on the rotation of the master axis. In Piecewise lock, the slave executes a Piecewise Profile when the master axis reaches a specified angle. Each of these modes is explained in detail below.

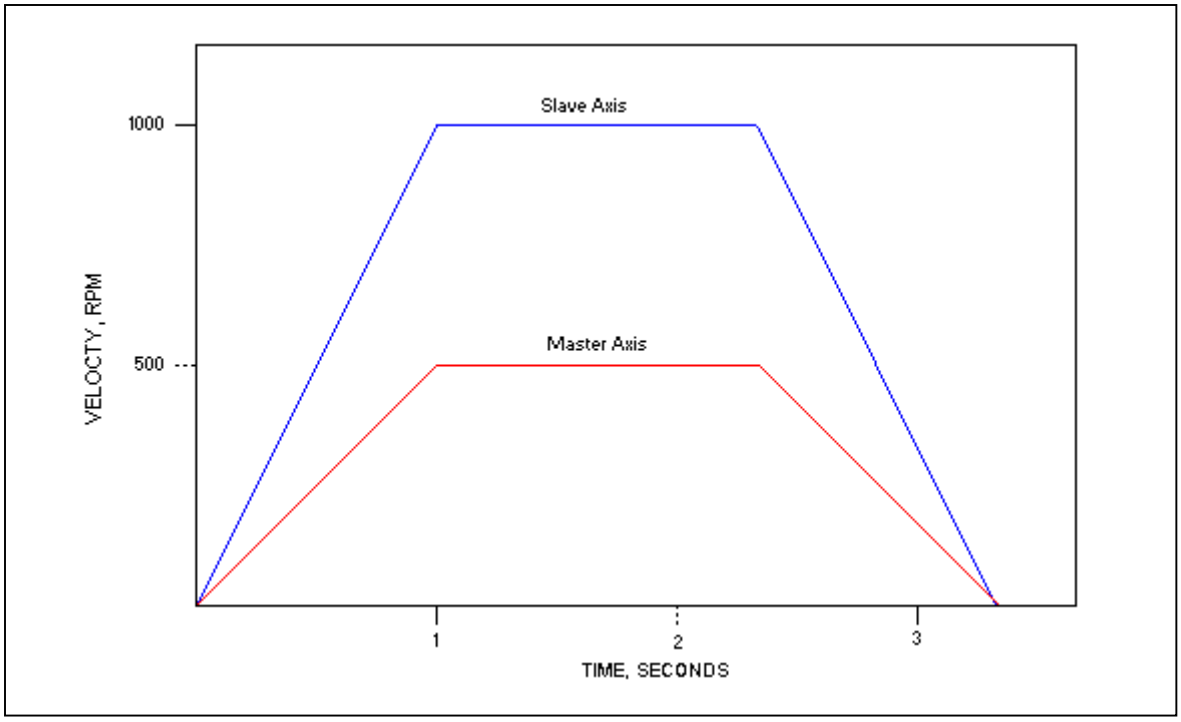


Figure 3.14 - Simple Lock Illustration

Simple Lock (Electronic Gearbox)

In simple lock mode, the position of the master axis is used directly to determine the proper slave axis position and speed. Figure 3.14 illustrates simple lock mode. In this example, the master axis and slave axis were positioned to zero before entering master slave lock. The slave axis has a ratio of 2.00.

NOTE: Note the motion profile of the slave in relation to the master. The slave axis always travels at twice the speed of the master. The distance traveled by the slave is twice that of the master.

Useful Facts About Simple Lock Mode

1. Ratios are treated as 12 bit fractions. To express a ratio of 1.0, the actual argument in the ratio instruction would be 4096.

2. Allowed values for ratio are from-32768 to +32767 bits. Positive ratios cause the slave axis to turn in the same direction as the master; negative ratios cause counter rotation.

3. When a program is started, the default ratio value for the slave axis is one-to-one (+4096 bits).

4. The recommended sequence for establishing master/slave lock is as follows:

   A. If necessary, position the master and/or slave to the desired locations.

   B. Establish the accel/decel rate of the slave. For certain lock methods, a high acceleration rate is needed to insure that the slave axis closely maintains the configured ratio between itself and the master axis.

   C. Issue a **set_map** instruction to start the information flow between the master and the slave.

   D. Lock the slave axis using the simple lock mode (mode 1).

   E. Set the desired ratio for the slave axis.

   F. Proceed with the desired motion for the master axis.

   Steps D through E can be used to lock on to a master axis already in motion.

5. A typical sequence for ending simple lock is:

   A. Unlock the slave axis using the **unlock** instruction.

   B. Determine that the slave axis has stopped by testing its **BUSY** status flag.
   C. Set the ratio for the slave axis to zero.

D. Terminate the master angle passing process by issuing a **set_map** instruction with no source axis.

6. Ratio may be changed at any time during simple lock mode. The slave axis will simply break lock, accelerate or decelerate at the currently programmed rate to match the master speed at the new ratio, and re-enter lock.

7. While in master/slave lock, a slave axis will ignore all motion instructions except forced deceleration. Issuing a **f_decel** or **unlock** instruction will break lock and cause the slave axis to stop.

Lock Methods For Simple Lock

There are four lock methods which can be used in simple lock applications. Each method and its function are outlined in Figure 3.15.

Lock Method	Description
1	Simple lock with acceleration limit. Slave tracks master position as long as currently set accel/decel rate is not exceeded.
2	Velocity Lock. Slave tracks master velocity, with accel/decel rate limit.
4	Simple lock with no acceleration limit.
6	Keyway to Keyway lock. Ratio is forced to be 1.0. On receipt of lock command, slave axis aligns its position transducer to match that of master. Accel/decel rate limit is used.

Figure 3.15 - Lock Methods for Simple Lock

Details for each of these lock methods are presented below.

Lock Method 1

Lock Method 1 provides the electronic equivalent of a gearbox. The electronic gearbox or ratio is driven by the master angle from one of the master angle buses. The master angle is processed by the previously specified **ratio** instruction and an offset is added, resulting in an effective master position used to drive the slave command position.

When the **lock** instruction is executed, with a Lock Method of 1, the axis controller calculates the instantaneous offset between the master angle processed by the ratio and the slave command position. Once the offset is calculated, master slave lock is accomplished. Once every 1 millisecond thereafter, the slave axis position is updated based on the new master position.

The slave motion is limited by the previously executed **set_ac_dc** instruction and further limited to a maximum speed of 5800 RPM. The limiting of the slave acceleration rate can be an advantage in systems that have rough or rapidly changing master speeds. The limited acceleration on the slave dampens the slave motor and smoothes out the operation.

The **ratio** instruction can be executed while in Lock Method. When the **ratio** instruction is executed, the slave controller stops executing the equation above and reverts to simply slewing at the **set_ac_dc** rate until the slave reaches the new target speed as if lock were in place. When the speed is matched, a new offset is calculated and the equation above is executed.

It is important to note that after a **ratio** instruction is executed, a new offset is used. The **JOGGING** status flag is on while the slave axis is changing from one speed to another. This flag goes off when the ratio lock equation starts executing.

The acceleration rate may be changed during the slew from one speed to another caused by a ratio change. A **set_ac_dc** prior to the **ratio** instruction will cause the slew acceleration rate to change.

The limiting of slave speed and acceleration results in some interesting situations. For example, if the master is turning when the **lock** instruction is executed, the slave motor will accelerate at the specified acceleration rate until the equation above is satisfied. In actual practice the slave motor will accelerate above what would appear to be the final speed in order to make up the angular phase difference lost during acceleration. The resulting motion profile is shown in Figure 3.24

Since the slave is limited in both speed and acceleration, a situation may occur where the slave can never reach final lock speed. The same situation can occur if the effective master is changing speed faster than the slave can change speed because of the acceleration limit. As a rule of thumb, the slave acceleration rate should be set to at least 5 times the expected rate of change of the effective master

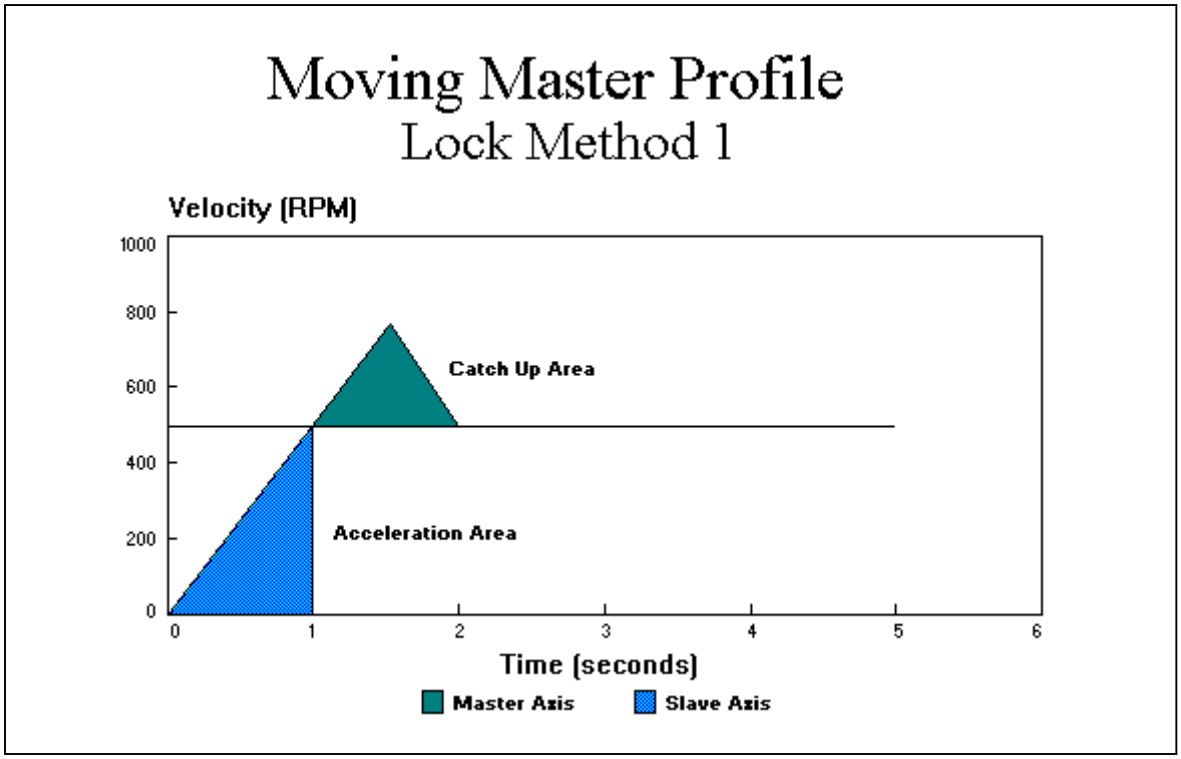


Figure 3.16 - Lock on a Moving Master

Lock Method 4

Lock Method 4 is identical to Lock Method 1, except that the slave is not limited by the specified slave acceleration rate. This means that during lock, the slave is commanded directly by the effective master position.

NOTE: Any perturbations or roughness in the master will be passed onto the slave with no rate limiting.

The **ratio** instruction can be executed during Lock Method 4. When a new **ratio** is executed, the slave controller breaks lock, slews to new lock speed at the specified acceleration rate and relocks using the above equation. The **JOGGING** status flag is on during the slew period. Note that after the slew period a new offset is calculated. The acceleration rate may be changed during slew by executing a **set_ac_dc** instruction prior to the **ratio** instruction.

Lock Method 6

Lock Method 6 allows the user to align the angle of the slave device with the angle of the master device. The ratio in Lock Method 6 is fixed at 1:1 and may not be changed. The slave controller acceleration rate is limited by the previously executed **set_ac_dc** instruction. The slave top speed is also limited to 5800 RPM.

When the **lock** instruction is executed, the slave controller executes the following equation every 1 millisecond.

Slave Angle = Master Angle

This equation is executed modulo 1 turn (4096 bits) of the master. This means that when the **lock** instruction is executed, the slave will move, at the acceleration rate specified, in the shortest direction to bring the master and slave absolute angles into alignment. Note that Lock Method 6 can cause movement when **lock** is executed even if the master is at rest.

The slave acceleration and speed are limited. Hunting can occur if the master rate of change is greater than the specified slave acceleration rate. The rule of thumb of having the slave acceleration rate at least 5 times the expected master rate of change is applicable. The limited acceleration rate can be used to smooth a master with perturbations (roughness).

Electronic Cams

The DeltaMax Controller provides a mode of operation that effectively emulates mechanical cams. In the electronic cam mode, slave axes follow digital cams based on the rotation of a master axis.

Electronic cams are tables of incremental motor moves. Each incremental value occupies one byte of memory in the DeltaMax and has an allowable range of -127 to +127. (A value of 128 is used to signal the end of the cam). As the master axis turns, its position is continually transmitted onto the data bus. The slave controller uses the current master position to determine the proper index into the electronic cam array. As the index moves through the cam table, the incremental distances are added together to form the proper slave axis position. Linear interpolation is performed between elements in the cam array. When the end of the cam table is reached, the process begins again at the beginning of the table.

The key DeltaMax macro instruction for electronic cam mode is the **cam_data** instruction.

This instruction defines the location of the cam within the axis controller memory. When the cam is defined within Macroprogram memory, this instruction will transmit the cam to the axis controller memory as well. The **cam_data** instruction will also establish the scaling factor for master position and cam elements. Master and data scaling are explained in the paragraphs below.

Master Scaling

Master scaling provides a mechanism to control the rate at which the slave axis processes its cam array relative to the rate at which the master axis turns.

In electronic cam mode, the master scale factor refers to the number of times the master position value is divided by 2 (i.e. shifted right) by the slave processor. For example, if a master scale factor of 12 were used, the slave processor would divide the master position value by 4096 (2^12). The slave processor then compares the least significant 12 bits of the scaled value to the previous scaled master position to determine if it has changed. If the value has changed, the cam array index is changed accordingly.

Figure 3.17 summarizes the effect of master scaling.

Master Scale Factor	Cam Array Advances Every
12	4096 master bits
11	2048 master bits
10	1024 master bits
9	512 master bits
8	256 master bits
7	128 master bits
6	64 master bits
5	32 master bits
4	16 master bits
3	8 master bits
2	4 master bits
1	2 master bits
0	1 master bits

Figure 3.17 - Master Scaling

Data Scaling

In order to insure that each element of an electronic cam is in the range of -127 to +127, it is often necessary to scale the cam data. This is accomplished by dividing the data by 2^n where n is selected to cause the largest element of the cam array to be less than -127 to +127.

For example, if the largest element in a cam array is 864, we would need to divide by 8 (2^3) to reduce the number below 128. The data scale factor in this case would be 3. Data scale factors from zero to 7 are allowed.

Important Notes Regarding Electronic Cams

The following items should be considered when using electronic cams (Note - These examples assume a positive ratio value is being used):

1. Positive cam data indicates to the slave axis that the slave should travel in the same direction as the master axis.

2. The cam data pointer will always start at the top of the cam table, unless the **set_cam_ptr** instruction and lock method 5 , 8 or 9 are used.

3. The cam data pointer will move toward the cam data table terminator when the master axis is traveling in the clockwise (CW) direction. The cam data pointer will move away from the cam data table terminator when the master axis is traveling in the counter-clockwise (CCW) direction.

Calculating Electronic Cams

There are several ways that electronic cam data can be transferred to the axis controller memory:

1. The cam data values can be precalculated and placed in cam tables using the instructions **begin_cam**, **cam** and **end_cam**. When the **cam_data** instruction is executed, the data defined in the cam table will be transferred to the axis controller along with the 'master scale' and 'data scale' values. The cam table terminator character (hex value 0x80) will be automatically transferred by the **cam_data** instruction, as well.

2. The cam data can be calculated by the program, or off-line by another computing device. This data can then be transferred to the axis memory using a series of **let_byte** instructions. The programmer in this instance is responsible for sending the cam table terminator character (hex value 0x80) as the last cam data table element. When the **cam_data** instruction is executed, only the 'master scale' and 'data scale' values are transferred to the axis controller, since it is expected that the data has been previously transmitted.

3. The cam data table can be calculated **by the axis controller** using the **calc_unit_cam** instruction. Using this approach, a data table defining the 'shape' of the profile is transmitted to the axis controller. The programmer then issues a **calc_unit_cam** instruction which will calculate the cam data table. When the **cam_data** instruction is executed, only the 'master scale' and 'data scale' values are transferred to the axis controller, since it is expected that the data has been previously calculated.

In applications where it might be desirable to compute an electronic cam based on a predetermined shape, such as a modified sine or trapezoidal type motion profile, the **calc_unit_cam** instruction can be used to greatly simplify this process.

The general process for using the **calc_unit_cam** instruction is as follows:

DEFINE THE MOTION SHAPE BY CREATING A 'UNIT CAM' TABLE

1. Either graphically or mathematically describe the motion profile desired. The profile should be a function of distance versus machine angle or other appropriate X-axis value.

2. Measure and tabulate the value of distance at 128 equally spaced X-axis intervals over the duration of the profile.

3. Normalize the distance values by dividing each tabulated value by the largest value in the table.

4. Scale each table value by multiplying it by 65,536 (2^16).

5. Add a zero element to the beginning of the table and replicate the last element in the table, so that the resulting table has 130 elements.

These steps result in a 'unit cam' table, which can then be used to generate cams of various lengths and number of elements.

TRANSFER THE 'UNIT CAM' TABLE TO THE CONTROLLER

6. Dimension data storage areas on the axis controller of interest as follows:

```
cam_area     dim     1,6750
table_area   dim     1,130
```

The labels 'cam_area' and 'table_area' may be named whatever you desire. The dimension numbers must be as shown. It would also be acceptable to have more than one 'cam_area' defined, as long as the total is equal to 6750 (this is 6750 4-byte elements totaling 27000 bytes). The axis controller expects to find the start of the 'table_area' at memory location 6750.

7. Load the 'unit cam' table into the axis controllers 'table_area' using the **let** instruction (the 130 entries in the 'unit cam' table are 4-byte values).

CALCULATE THE CAM

8. Determine the total distance to be represented by the cam, and the total number of cam elements required.

9. Issue the appropriate **calc_unit_cam** instruction. While the axis controller is busy calculating the cam, its status flag for **CALCULATING PW PROFILE** is set.

NOTE: It is not necessary to issue a **drive_on** instruction before executing the **calc_unit_cam** instruction.

10. Insert an end of cam character (hex value 0x80) at the appropriate place in the cam array.

11. Issue a **cam_data** instruction so that the appropriate 'master scale' and 'data scale' are used.

At this point, the axis controller contains a valid electronic cam. Processing can now follow normal cam procedures.

The instruction format for **calc_unit_cam** is as follows:

```
label     calc_unit_cam     axis#,distance,elements
```

where 'axis#' is always 1, 'distance' is the total distance in counts to be traveled and 'elements' is the size of the cam to be generated.

The following example illustrates the **calc_unit_cam** instruction and the steps taken to load the predetermined 'unit cam' table named 'trap_1_3' into the axis controllers memory:

Program Example

```
                declare         off
!
! ----------  FLAG DEFINITION  ----------
!
CALC            equ             74
DOWN            equ             77
BUSY            equ             78
LOCK            equ             79
AXIS_1          equ             1
BUSY_1          equ             (AXIS_1*16)+BUSY
DOWN_1          equ             (AXIS_1*16)+DOWN
CALC_1          equ             (AXIS_1*16)+CALC
LOCK_1          equ             (AXIS_1*16)+LOCK

cam             dim             1,6750
table           dim             1,130

!
! ----------  TABLE FOR UNIT TRAPEZOIDAL CURVE 1/3,1/3,1/3  ----------
!
trap_1_3        begin_data
                data            0,9,36,81,144,225,324
                data            441,576,729,900,1089,1296,1521
                data            1764,2025,2304,2601,2916,3249,3600
                data            3969,4356,4761,5184,5625,6084,6561
                data            7056,7569,8100,8649,9216,9801,10404
                data            11025,11664,12321,12996,13689,14400,15129
                data            15876,16640,17408,18176,18944,19712,20480
                data            21248,22016,22784,23552,24320,25088,25856
                data            26624,27392,28160,28928,29696,30464,31232
                data            32000,32768,33536,34304,35072,35840,36608
                data            37376,38144,38912,39680,40448,41216,41984
                data            42752,43520,44288,45056,45824,46592,47360
                data            48128,48896,49660,50407,51136,51847,52540
                data            53215,53872,54511,55132,55735,56320,56887
                data            57436,57967,58480,58975,59452,59911,60352
                data            60775,61180,61567,61936,62287,62620,62935
                data            63232,63511,63772,64015,64240,64447,64636
                data            64807,64960,65095,65212,65311,65392,65455
                data            65500,65527,65536,65536
                end_data
```

```
!
! -----   LOAD THE UNIT CAM TABLE INTO AXIS #1 STARTING AT LOC 6750  -----
!
load_table      let             ctr=0
load_loop       let             t=trap_1_3[ctr]
                let             table[ctr]=t
                let             ctr=ctr+1
                if              ctr<130,load_loop


!
! -----   CALCULATE THE CAM BASED ON UNIT CAM TABLE   -----
!
                let             distance=40960
                let             elements=2048
                let             start_of_cam=0

                calc_unit_cam   AXIS_1,distance,elements,start_of_cam
calc_busy       if_stat_on      CALC_1,calc_busy


!
! -----   CALCULATE THE CAM SUM FOR THE CAM   -----
!
                let             end_of_cam=elements-1

                calc_cam_sum    AXIS_1,start_of_cam,end_of_cam
sum_busy        if_stat_on      CALC_1,sum_busy


!
! -----   GET SUM, SHOULD EQUAL REQUESTED CAM DISTANCE   -----
!
                get_cam_sum     AXIS_1,sum
                if              sum<>distance,calc_err


!
! -----   INDICATE THE CAM TO USE, MASTER SCALE AND DATA SCALE   -----
!
                cam_data        AXIS_1,cam,5,0
                ratio           AXIS_1,4096
```

Electronic Cam Lock Methods

Lock Method	Description
0	Cam lock at beginning of cam data table.
5	Cam lock at current cam pointer position. Used in conjunction with the **set_cam_ptr** instruction to lock at other than the beginning of the cam data table.
8	Cam Lock at current cam pointer position once the specified master angle is crossed. (See **set_trig_cam).**
9	Same as Lock type 8 except execution of the cam is terminated at the last element in the cam.

Figure 3.18 - Lock Methods for Cam Lock

A detailed explanation of each of these lock methods follows.

Lock Method 0

When the **lock** instruction is executed with a lock method of 0, the axis controller puts the cam pointer at the very top of the cam array, creates an instantaneous offset between the absolute master angle (on the selected bus) and 0.00 effective master, then locks the axis to the master. Once per millisecond after the **lock** instruction, the following equation is executed to drive the cam pointer.

$$\frac{\text{(Absolute master angle * ratio)}}{\text{(2\^master scale)}} + \text{offset} = \text{effective master}$$

As this equation executes, the integer part of the effective master causes the cam pointer to move and the fractional part is used to linearly interpolate cam elements. The offset value in the equation is set when the **lock** instruction is executed to yield a 0.00 effective master and is modified each time the cam pointer wraps around to maintain the cam pointer within the cam data array.

If the ratio is changed during **lock** in Lock Method 0, the offset is instantaneously changed to yield the same effective master.

Changing the ratio on the fly during Lock Method 0 does not cause a jump in the cam pointer, but does cause a change in rate or direction of cam pointer movement. Since the slave speed and acceleration rate are not limited, the slave servo may change speed abruptly. Caution should be used when changing the **ratio** during cam execution.

During Lock Method 0, the pointer position in the cam data table can be captured using the **get_cam_ptr** instruction.

Lock Method 5

Lock Method 5 is identical to Lock Method 0, except that the cam pointer is not moved to the top of the cam array when the **lock** instruction is executed. The cam pointer is started at the beginning of the cam element wherever the cam pointer was previously left (from previous cam execution) or a **set_cam_ptr** instruction execution.

Lock Method 8

Lock Method 8 causes execution of a cam to be triggered when a specified master angle is crossed. Execution begins at the current cam pointer (see **set_cam_ptr**). The master angle to be used as the trigger will be defined using the **set_trig_cam** instruction.

Lock Method 9

Lock Method 9 is identical to lock method 8 except execution of the cam is terminated when the last element in the cam table is executed.

Sample Electronic Cam Application

An application requires a slave axis to follow a motion described by the mechanical cam shown in Figure 3.19. It has been determined that 16 cam data points will provide sufficient resolution for this application. The master axis is known to make 4 complete rotations for one cycle of the cam.

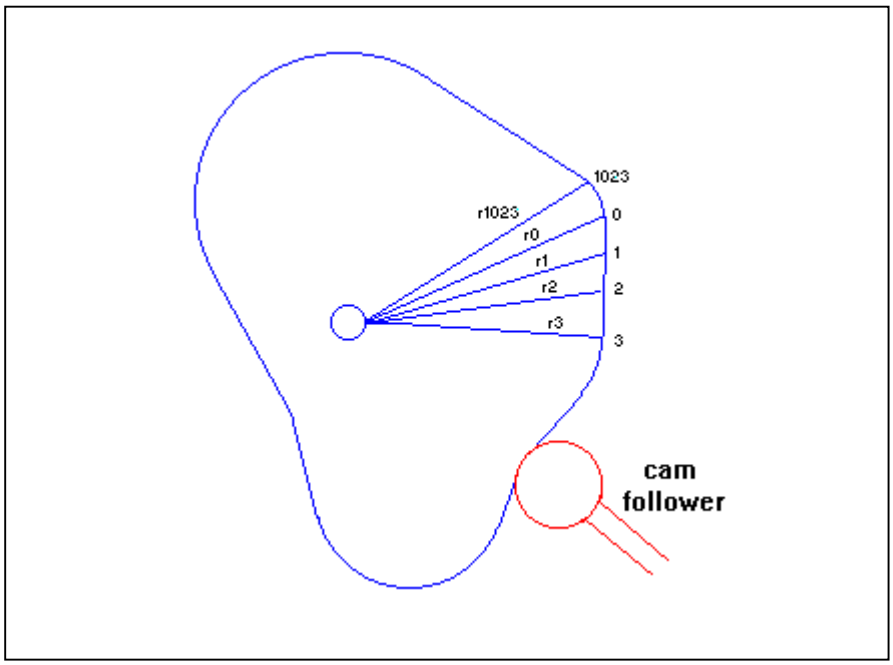


Figure 3.19 - Mechanical Cam

To implement this application, the following procedure could be used:

1. Measure the radius of the cam at each of 16 equal angle increments around the cam.

2. Convert data to incremental form by subtracting each data point from its successor.

3. Convert the data from inches to position transducer units by applying the proper scaling factor (4096 in this case.)

4. Examine each cam array to find the largest value. Use this value to determine the data scale factor needed to scale the data to the range of -127 to +127.

5. Divide each cam array element by the appropriate value (2^n, where n is the data scale factor).

6. Use a master scale factor of 10 (1024 master bits = 1 cam element).

The resulting arrays can then be entered into a program to produce the desired motion. Figure 3.20 shows a graphic representation of the resulting cam motion.

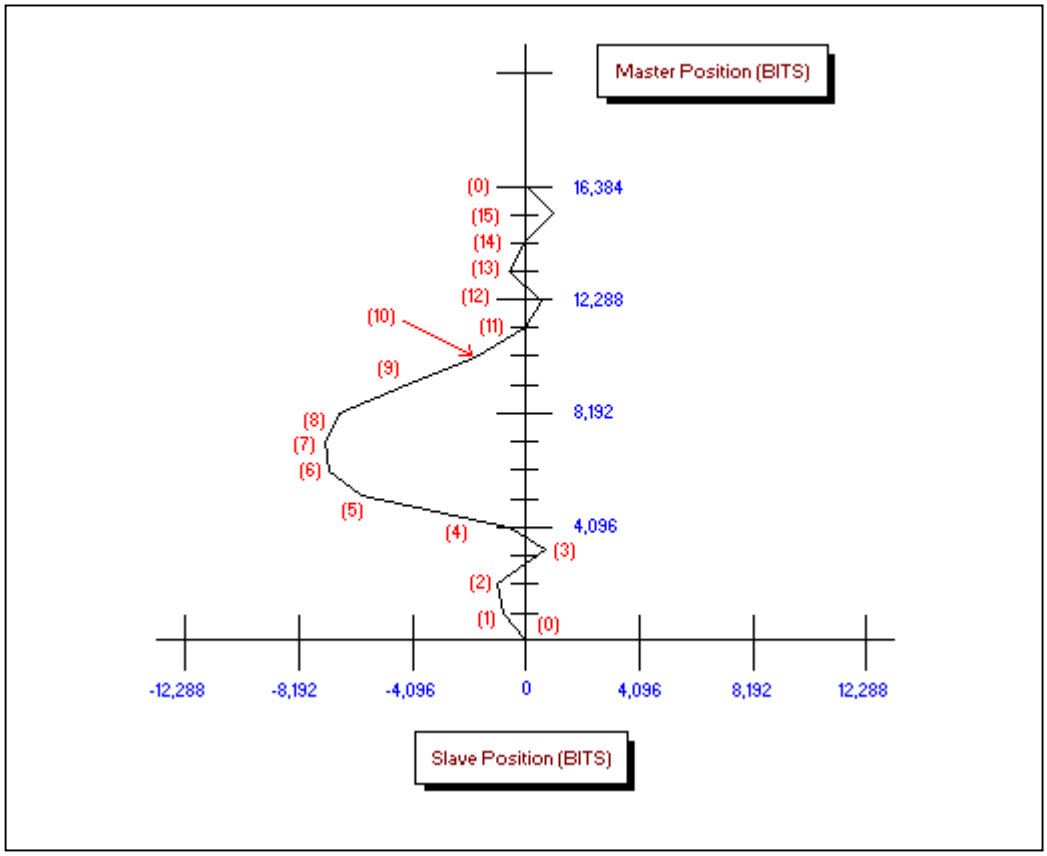


Figure 3.20 - Resulting Master/Slave Motion

Piecewise Lock

Piecewise Lock, Lock Method 3, provides a means of triggering execution of a Piecewise Profile in a slave axis based on the master reaching a certain angular position. Refer to the Section on Piecewise Profiles for more details

To establish Piecewise Lock, follow the steps outlined below.

1. Build the profile as described in the Piecewise Profile Section.

2. Establish appropriate master angle passing.

3. Issue the **prep_profile** command to transfer the profile data to the slave axis card.

4. Set up the trigger angle with the **set_trig_pw** instruction.

5. Issue the **lock** command with Lock Method set to 3.

When the specified master angle position is reached, the Piecewise profile will be executed.

Master Angle Bus

The master angle bus is a high speed data highway which links the master/slave functions in a DeltaMax Controller. Each controller has two independent buses, designated as Master Bus A and Master Bus B. The **set_map** program instruction provides a means for the programmer to specify which controller talks as a master on a given bus and which controller(s) listen as slaves.

There can only be one controller as master on a bus. Any controller may be configured as a slave. One slave controller may not be configured to listen to both masters.

The **set_map** instruction uses a 4 byte word (32 bits) to define the total configuration. In the 32 bit variable, two (2) bytes are used for each bus. One byte configures which controller is the master and the other byte configures which controllers are the slaves. The **set_bit** instruction provides a convenient means of setting the proper bits in the **set_map** variable.

Figure 3.30 shows the configuration of the **set_map** variable for the DeltaMax.

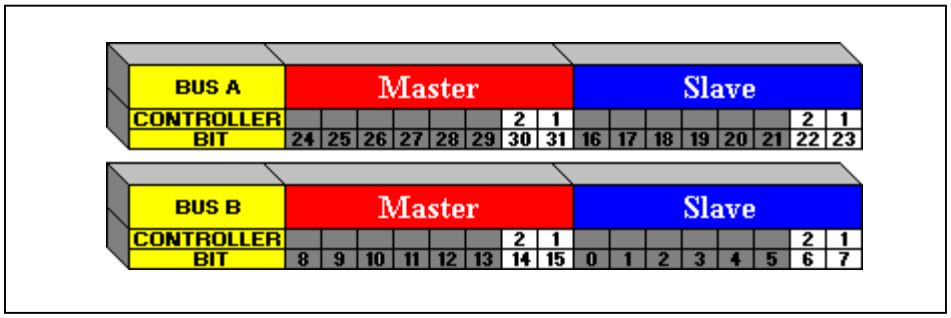


Figure 3.21 - DeltaMax Master Angle Bus Configuration

Consider an example where the pseudo is to be the master axis, and controller 1 is to be a slave. Master Angle Bus A will be used for the data path. The following program instructions could be used to configure the master angle bus:

Program Example

```
!
! -----  DEFINE BITS TO CONFIGURE THE MASTER ANGLE BUS STRUCTURE  ------
!
MASTER_2    equ         30                  Controller 2 talker on Bus A
SLAVE_1     equ         23                  Controller 1 listen on Bus A

!
! -------  INITIALIZE THE MASTER ANGLE BUSES - NO ACTIVITY  -------
!
                let         mapvar=0
                set_map     mapvar

!
! -------  TURN ON THE NECESSARY BITS IN THE VARIABLE mapvar -------
!
                set_bit     MASTER_2,mapvar
                set_bit     SLAVE_1,mapvar

!
! -------  CONFIGURE THE BUS STRUCTURE USING THE VARIABLE mapvar -------
!
                set_map     mapvar
```

Master Angle Bus Cautions

It is important to note that the data being transmitted on a master angle bus is the current master **position transducer reading**. Program instructions such as **set_local** or **set_offset** DO NOT affect the data being transmitted on a master angle bus, but DO impact the logical position of an axis. Thus, it is possible to read a position from a master controller using a **get_pos** instruction and get a position value which does not match the value being transmitted on the master angle bus.

Fiber Optic Network

Each DeltaMax Controller has a single fiber optic communications port. This port contains a single receiver. This provides a link between an MSC-850/MCF-850 or RFC-100 or an EFC-100 and a DeltaMax. This port must be configured using the **set_mcf** instruction. Figure 3.22 shows the bit assignments used to define the network data paths.

Figure 3.22 - DeltaMax Multi-Function Controller Configuration

Program Example - DeltaMax Controller #1

```
!
! ------- BITS USED TO CONFIGURE THE MASTER ANGLE BUS STRUCTURE  -------
!
MASTER_2        equ     30                      Controller 2 talker on Bus A
SLAVE_1         equ     23                      Controller 1 listen on Bus A
FO_SOURCE_A     equ     8                       Fiber Optics is the source for MAB A
MCF             equ     2                       MCF location

!
! ------  DEFINE THE MULTI-FUNCTION CONTROLLER CARD  ------
!
                let     mcfvar=0
                set_bit FO_SOURCE_A,mcfvar
                set_mcf MCF,mcfvar

!
! ------  INITIALIZE THE MASTER ANGLE BUSES - NO ACTIVITY  ------
!
                let     mapvar=0
                set_map mapvar

!
! ------  TURN ON THE NECESSARY BITS IN THE VARIABLE mapvar ------
!
                set_bit MASTER_2,mapvar
                set_bit SLAVE_1,mapvar

!
! ------  CONFIGURE THE BUS STRUCTURE USING THE VARIABLE mapvar ------
!
                set_map mapvar
```

Master/Slave Instructions

Instruction	Format		
calc_cam_sum	*label*	**calc_cam_sum**	controller#,start_element,ending_element
calc_unit_cam	*label*	**calc_unit_cam**	controller#,distance,#elements,start_element
cam_data	*label*	**cam_data**	controller#,label,ms,ds
data_scale	*label*	**data_scale**	controller#,dscale
get_cam_cnt	*label*	**get_cam_cnt**	controller#,value
get_cam_end	*label*	**get_cam_end**	controller#,value
get_cam_ptr	*label*	**get_cam_ptr**	controller#,value
get_cam_strt	*label*	**get_cam_strt**	controller#,value
get_map	*label*	**get_map**	value
get_map_stat	*label*	**get_map_stat**	value
get_mcf	*label*	**get_mcf**	controller#,value
get_mcf_stat	*label*	**get_mcf_stat**	controller#,value
incr_com	*label*	**incr_com**	controller#,bits,interrupts
lock	*label*	**lock**	controller#,lock#
master_scale	*label*	**master_scale**	controller#,mscale
ratio	*label*	**ratio**	controller#,ratio
set_cam_ptr	*label*	**set_cam_ptr**	controller#,value
set_map	*label*	**set_map**	value
set_mcf	*label*	**set_mcf**	controller#,value
set_trig_cam	*label*	**set_trig_cam**	controller#,master_angle
set_trig_pw	*label*	**set_trig_pw**	controller#,master_angle
switch_cam	*label*	**switch_cam**	controller#,start_element
unlock	*label*	**unlock**	controller#,mode#

Figure 3.23 - Master/Slave Instruction Summary

Programmable Limit Switches

DeltaMax Controllers provide a means of switching output modules on and off in relationship to the position of a master axis. This function, called Programmable Limit Switch (PLS), is similar to that provided by mechanical cam or drum switches. The DeltaMax Controller provides basic PLS capabilities, which are described in detail below.

DeltaMax PLS Functions

One of the features of the DeltaMax Controller is the Programmable Limit Switch feature. There are 16 PLS output flags available. These flags can be switched on and off, depending on the position of the master angle being used to drive the PLS function. There are 8 hardware modules and 8 software flags. In the DeltaMax, the 8 hardware modules are associated with 8 physical output modules which correspond to flags 16 to 23.. The eight software outputs correspond to flags 24 to 31 and can be used to trigger software interrupts. The state of all 16 flags can be monitored using the **get_pls_out** instruction.

The users' program specifies set points (angles) of the master angle bus where the 16 flags should be turned ON and OFF. During program execution, this angular position is monitored and the outputs are turned ON and OFF according to the programmed set points.

Programming using PLS's

Before attempting to write a program using the Programmable Limit Switch feature of the DeltaMax, it is important to understand the following:

1. Only one ON and OFF angle may be programmed for each PLS output. When an individual PLS module is programmed more than once using the **set_pls_ang** instruction, the new information regarding that module will replace the current information.

2. The ON and OFF set points are interpreted assuming a clockwise direction of rotation. For example, ON at 1000, OFF at 2000, means that the PLS output will be enabled whenever the master angle position is between 1000 and 1999 bits. However, ON at 2000, OFF at 1000, means that the PLS output will be disabled between 1000 and 1999 bits and enabled for the remainder of the master rotation.

3. Individual PLS modules can be disabled and enabled using the **set_pls_mask** instruction. Using this instruction, the PLS modules do not need to be reprogrammed.

4. Once a PLS module has been defined, it can only be cleared by using the **clr_pls** instruction.

Processing of programs using PLS's

When the controller receives a **set_pls_ang** instruction, it searches the programmed data for an existing record for the specified output. If a matching record is found, it will be replaced by the new data.

While this instruction is being processed, the CALCULATION IN PROGRESS flag of the axis controller will be activated. The program must verify that this flag is no longer active before executing subsequent **set_pls_ang** instructions. The axis processor will ignore any **set_pls_ang** instructions issued while the CALCULATION IN PROGRESS flag is activated.

Execution of programs using PLS's

The controller monitors data from a master angle bus and turns ON and OFF the outputs at programmed set points. The appropriate bit in the PLS mask value must be set in order for that PLS to be active. A bit which is clear in the PLS mask value indicates a PLS that will not be active, although it remains defined. Once a PLS is defined, that module remains a PLS, even if its' mask bit is cleared. It cannot be used as an output module until a **clr_pls** instruction is executed.

When a **pls_mask** instruction is executed and a PLS is enabled, the PLS module will immediately be updated (set or cleared). When a **pls_mask** instruction is executed and a PLS is disabled, the PLS module will be left in its current state.

NOTE: The angular position can come from one of several sources, as programmed by the **set_mcf** instruction.

DeltaMax

A. Pseudo Axis
B. Master Angle Bus A
C. Master Angle Bus B
D. Fiber Optic Receiver

When the DeltaMax is powered down, the PLS (Programmable Limit Switch) configuration is lost. The PLS modules must be programmed again on power up.

Programmable Limit Switch Instructions

Instruction	Format		
clr_pls	*label*	**clr_pls**	controller#,variable
get_angle	*label*	**get_angle**	controller#,variable
get_for_ang	*label*	**get_for_ang**	controller#,channel#,variable
get_pls_ang	*label*	**get_pls_ang**	controller#,pls_angle
get_pls_mask	*label*	**get_pls_mask**	controller#,variable
get_pls_out	*label*	**get_pls_out**	controller#,variable
preset	*label*	**preset**	controller#,variable
set_gl_ccw	*label*	**set_gl_ccw**	controller#
set_gl_cw	*label*	**set_gl_cw**	controller#
set_local	*label*	**set_local**	controller#
set_mcf	*label*	**set_mcf**	controller#,variable
set_pls_ang	*label*	**set_pls_ang**	controller#,on,off,module#
set_pls_cnt	*label*	**set_pls_cnt**	controller#,count
set_pls_mask	*label*	**set_pls_mask**	controller#,variable
set_pls_time	*label*	**set_pls_time**	controller#,time,module#

Figure 3.24 - Programmable Limit Switch Instructions

Interrupts

It is often necessary for your application program to respond quickly to an external event, such as a switch closure or an operator input. The DeltaMax programming language provides two methods for responding to this type of event.

Software Interrupts provide a means for a program to respond to changes in state of any DeltaMax flag. Hardware Interrupts provide a means for an axis controller to perform a preprogrammed task immediately on receipt of an input signal.

Software Interrupts

The Software Interrupts feature of the DeltaMax provides for automatic program response to the change in state of any DeltaMax flag (I/O, timer, motor status or user).

This feature functions by allowing the programmer to associate the change of state of a flag with a program subroutine. Whenever the specified change occurs, the DeltaMax makes a note of what it was doing, and then transfers control to the associated subroutine just as though a **gosub** instruction was performed. When the subroutine is completed, control passes back to the interrupted operation.

In the DeltaMax, Software Interrupts are checked between the execution of program instructions. Therefore, program activities such as downloading large cams from program memory to axis controller memory can hold off the recognition of software interrupts.

Software Interrupt processing provides up to 32 prioritized interrupts ranging from priority 0 (highest) to 31 (lowest). If more than one interrupt occurs at the same time, the one with the LOWEST interrupt number (highest priority) will be recognized first. The user may disable all or selected software interrupts during portions of the program which should not be "interrupted".

Software Interrupt processing is edge triggered -- the specified subroutine will be executed only on the leading or trailing edge of the transition, depending on the type of interrupt specified. For example, if a software interrupt is specified to occur whenever input module number 1 is ON and software interrupt processing is enabled while input module number 1 is already ON, no software interrupt will occur. The interrupt will only occur when the transition from the OFF to the ON state occurs.

Hardware Interrupts

In some cases, it is desirable for an axis controller to respond very rapidly to an external event. For these instances, DeltaMax Controllers provide a feature called **Hardware Interrupts**.

The axis controller is directly connected to the corresponding I/O Module Controller slot. For example,

Controller #1 is linked to I/O-1 (flag 0)

Hardware Interrupts can be processed much faster than conventional Input/Output handling since the Main Processor distributes the desired task to the Axis Controller before the interrupt occurs rather than waiting for the Input module to be activated, then distributing the task or instruction. Because of this, Hardware Interrupts are capable of 1 millisecond response times.

The following is a list of the tasks or instructions which can be executed when the corresponding hardware interrupt signal is detected:

trap_pos	**lock**
over_draw	**ratio**
index	**exec_profile**
position	**f_decel**

Each instruction above is used with the Hardware Interrupt feature by issuing an **enable_hwi** instruction, followed by the instruction to be executed when the interrupt is detected.

The following example would instruct controller #1 to index 2048 bits when input module I/O-1 changes from a low to high state:

```
label     enable_hwi
          index       1,2048
```

Execution of this pair of instructions causes the Main Processor to instruct the appropriate controller card that it should now monitor its Hardware Interrupt signal and perform the designated operation when the interrupt signal is activated. The axis controller responds by enabling its **BUSY** and **HARDWARE INTERRUPT ARMED** flags. After the controller detects the interrupt and completes the task, its **BUSY** and **HARDWARE INTERRUPT ARMED** flags will be disabled. It is the programmers responsibility to issue this pair of instructions again if it is desired to perform the task again.

The programmer can cancel an armed task with the instruction **disable_hwi**. On receipt of the **disable_hwi** instruction, the Controller's **BUSY** and **HARDWARE INTERRUPT ARMED** status flags will be turned off.

Only 1 task can be executed per interrupt. If more than 1 task is issued, the most recent will be used.

Interrupt Instructions

Instruction	Format		
clr_swi	*label*	**clr_swi**	interrupt#
clr_all_swi	*label*	**clr_all_swi**	
disable_swi	*label*	**disable_swi**	
disable_hwi	*label*	**disable_hwi**	controller#
enable_hwi	*label*	**enable_hwi**	
enable_swi	*label*	**enable_swi**	
f_decel	*label*	**f_decel**	controller#
get_trap_pos	*label*	**get_trap_pos**	controller#,variable
over_draw	*label*	**over_draw**	controller#,speed,limit,distance
set_swi_mask	*label*	**set_swi_mask**	variable
swi_if_off	*label*	**swi_if_off**	interrupt#,flag,sub_label
swi_if_on	*label*	**swi_if_on**	interrupt#,flag,sub_label
trap_pos	*label*	**trap_pos**	controller#

Figure 3.25 - Software and Hardware Interrupt Instructions

Extended Memory Operations

The DeltaMax provides one type of extended memory - volatile RAM. Separated into two distinct areas for controller 1 and 2.

Extended RAM

Each extended memory area has 28K bytes of volatile data memory. This memory can be used to create very high resolution electronic cam data tables or piecewise profiles.

Extended RAM Programming

To create a data array in axis controller memory, a special form of the **dim** instruction is used:

SYNTAX:

label **dim** controller#,words

PARAMETERS:

label	Name assigned to block of memory (array).
controller#	Axis ID #.
words	Length of data memory allocated to the label. Allocated in 4 byte (32 bit) words.

Once this special form of the **dim** instruction has been executed, it is not necessary to reference this area in memory by Axis ID#. Values in these arrays are accessed using the **let_byte** and **let** instructions in the same manner as any standard array.

WARNING:
Piecewise Profiles and the conventional form of the **cam_data** instruction use the same memory from the start (location 0) of the 28K volatile RAM area. It is the responsibility of the programmer to ensure that arrays declared with the extended form of the **dim** statement do not overlap with Piecewise Profiles or conventional cams.

Extended Memory Limitations

1. The **prep_profile** instruction for Piecewise Profiles may not reference data stored in axis controller memory. Data for this type of profile must reside in main memory.

2. The **cam_data** instruction can only reference data residing in the master controller or in axis controller memory which is the target of the **cam_data** instruction.

3. Depending upon the location of the data array, the **cam_data** instruction executes differently. If the array is in the master controller's memory, the **cam_data** instruction transmits the array to the specified axis controller, establishes pointers to the beginning and end of the cam array, and defines both the master scale and data scale factors. If the array is in the axis controller's memory, the **cam_data** instruction only establishes pointers to the beginning and end of the array, and the master and data scale factors. No data transfer is executed.

Analog Input/Output

DeltaMax

The DeltaMax has two analog inputs and one analog output channel. These channels might be used in applications like a joystick interface, controlling DC motor drives, or monitoring analog sensors.

Capabilities of Analog Input/Output

Data range 12 bit conversion over a range of 0 to +5 volts is provided. The data range is 0 (0 volts) to +1023 (+5 volts).

Offset Any of the 3 channels may be offset. The offset value is added to the real channel value at the time of conversion. The channel offsets range from -1023 to +1023. The default value is 0.

Slew rate An analog “rate of change” limit may be set for each channel. This rate of change limit is "calibrated" in bits per 10 milliseconds. The default value is +1023.

Analog Controller Functional Description

Analog processing uses a 10 millisecond update cycle in which the analog input channels and analog output channel are updated.

INPUT channels are read-only. Each input channel is read, and the raw voltage signal is added to the corresponding channel offset value. The result of this addition is compared to the previous reading for that channel, and if necessary, limited by the slew rate limit currently in effect.

OUTPUT channels are updated using the same algorithm as INPUT channels. That is, the offset is added and then the rate of change limit is applied. This limited result is then output.

Power-Up States for Analog I/O Channels

On power up, or after a **RESET** command from the MacroPro II Analyzer, the analog I/O channels are in the following state:

1. Offsets are set to zero (0).

2. Slew rate limits are set to 1023.

3. Output channel values are set to zero.

Analog Instructions

Instruction	Format		
analog_in	*label*	**analog_in**	controller#,ch#,value
analog_out	*label*	**analog_out**	controller#,ch#,output
analog_rt	*label*	**analog_rt**	controller#,ch#,value
analog_zo	*label*	**analog_zo**	controller#,ch#,value

Figure 3.26 - Analog Instructions

User Serial Ports

The DeltaMax family of controllers support a number of serial communication ports. The following is a summary of the ports available for each controller and a short description of each:

PORT NUMBER	DESCRIPTION
Port 1	RS-232C Executive serial port.
Port 2	RS-232C Executive serial port ***or*** User programmable RS-232 port.

Port 1 is an RS-232C serial port that only supports the DeltaMax Packet Protocol method of communication. This port is typically used by the MacroPro II Development System for the loading and testing of DeltaMax programs.

Port 2 is a user programmable port suitable for use with a variety of computer displays, hand-held terminals, strip displays, printers, data entry terminals, etc..

Both Port 1 and Port 2 default to RS-232C Executive serial port with a baud rate of 19,200 and will remain in this configuration until a **port_set** instruction is executed. The **port_set** instruction can only be applied to port Port 2.

Serial Port Initialization

Before the user programmable port (Port 2) can be used, it is necessary to configure the port for proper use. The **port_set** instruction is used for this purpose. The format of this instruction is:

label **port_set** **port,baud,protocol**

The **port** number can only be 1. The **baud** rate can be 110, 300, 600, 1200, 2400, 4800, 9600, or 19200. The **protocol** variable is used to select the proper combination of parity, stop bits and XON/XOFF selection, according to Figure 3.39.

Protocol	Description
0	1 stop bit, no parity, XON/XOFF disabled
1	2 stop bits, no parity, XON/XOFF disabled
2	1 stop bit, odd parity, XON/XOFF disabled
3	2 stop bits, odd parity, XON/XOFF disabled
4	1 stop bit, even parity, XON/XOFF disabled
5	2 stop bits, even parity, XON/XOFF disabled
6	CUSTOM
7	RESERVED
8	1 stop bit, no parity, XON/XOFF enabled
9	2 stop bits, no parity, XON/XOFF enabled
10	1 stop bit, odd parity, XON/XOFF enabled
11	2 stop bits, odd parity, XON/XOFF enabled
12	1 stop bit, even parity, XON/XOFF enabled
13	2 stop bits, even parity, XON/XOFF enabled

Figure 3.27 - Communication Protocol Selection

Once a **port_set** instruction has been executed, it remains in effect until another is issued or power is cycled to the DeltaMax.

Once a serial port has been initialized, **print**, **print_num**, and **input** instructions may be issued.

Important Notes Regarding Serial Ports

The following factors should be considered when using the DeltaMax serial ports.

1. Serial port instructions have been implemented so that program execution does not need to be delayed while waiting for characters to be transmitted or received. This leads to the following considerations:

   a. **Print** and **print_num** instructions are queued up to be executed by a separate task. Several milliseconds may pass between the execution of the instruction and the actual transmission of characters. The program sequence:

   ```
   label       let         x=10
               print_num   4,0,x
               let         x=25
   ```

   could result in "25" being sent out the serial port instead of "10".
   It may be necessary to use temporary variables to yield the desired result.

   b. The print queue of the operating system has room for a limited number of entries. It is possible for the user to overrun this queue by rapidly issuing print commands with no time delay between them. If your application will issue several print commands in succession, it may be necessary to implement a software delay between print commands.

   c. The **input** instruction takes precedence over the **print** and **print_num** instructions. The program sequence:

   ```
   NULL          text      ""                          null
                           string
   MSG_PROMPT    text      "Please Enter Speed: "
                 print     MSG_PROMPT
                 input     NULL,4,0,x,in_done
   ```

   may result in the **input** instruction executing before **MSG_PROMPT** is sent out the port. A programmed wait of approximately 30 milliseconds between the **print** and the **input** statement may be necessary to achieve the desired result.

2. There are two special cases of the **input** command:

   a. The first parameter of the **input** instruction is the address of a text string which is sent to the serial port as a prompt. If the prompt string exists, then the current value of the variable being input is sent to the port immediately following the prompt string. If the prompt string is a null string, i.e. consists of a single null character, then no prompt is displayed, and the current value of the variable is NOT sent to the port.

b. A special form of the **input** command is provided to handle single character input. In this form, the prompt string is a null string and the length and width parameters are zero. In this situation, the decimal value of the next character received via the serial port will be placed in the input variable.

Serial Instructions

Instruction	Format		
if_char	*label*	**if_char**	port#,address_label
if_no_char	*label*	**if_no_char**	port#,address_label
input	*label*	**input**	textlabel,len,dec,variable,userflag
port_set	*label*	**port_set**	port#,baud,protocol
print	*label*	**print**	textlabel
print_num	*label*	**print_num**	len,dec,var
stop_input	*label*	**stop_input**	

Figure 3.28 - Serial Port Instructions

MACROPRO II™

FOR WINDOWS™

Section 4 - DeltaMax DeviceNet

INDUSTRIAL INDEXING SYSTEMS, INC.

SECTION 4 - DELTAMAX DEVICENET

Table of Contents

This page is intentionally left blank.

Introduction

DeviceNet is a low-cost communications link to connect industrial devices (such as limit switches, photoelectric switches, valve manifolds, motor starters, operator interfaces, ect..) as well as control devices to a network. The Open DeviceNet Vendor Association Publishes a DeviceNet Product Catalog three or four times a year for a current product listing of DeviceNet compatiable devices.

DeviceNet is a trademark of the ODVA Open DeviceNet Vendor Association, inc.

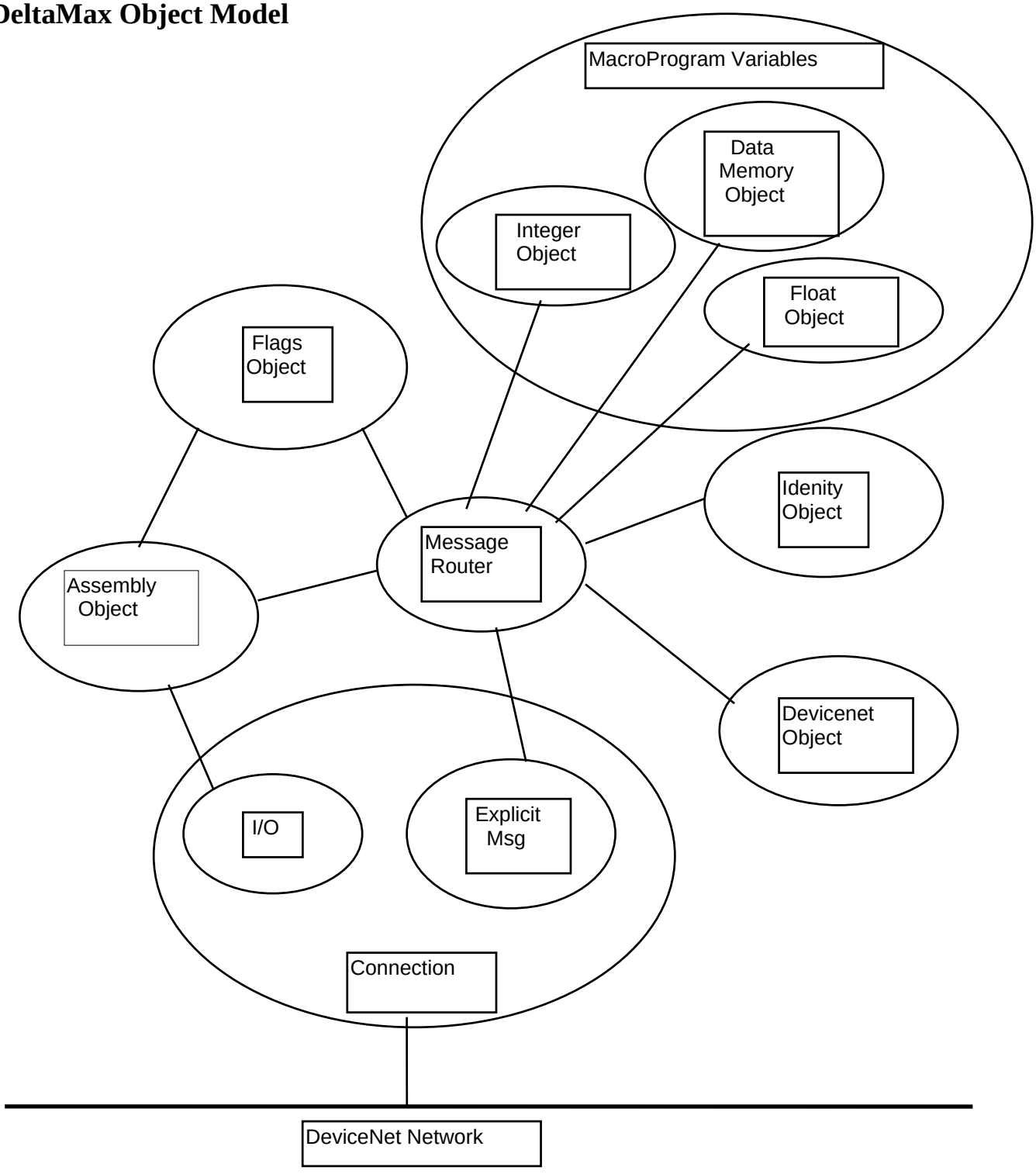
8222 Wiles Road	Phone: (305)340-5412
Suite 287	Fax:(305)340-5413
Coral Springs , FL 33067	http://www.industry.net/ODVA

IIS Industrial Indexing Systems Inc. is a member of ODVA. IIS maintains a Vendor ID with ODVA of 89.

IIS has given the DeltaMax the following:

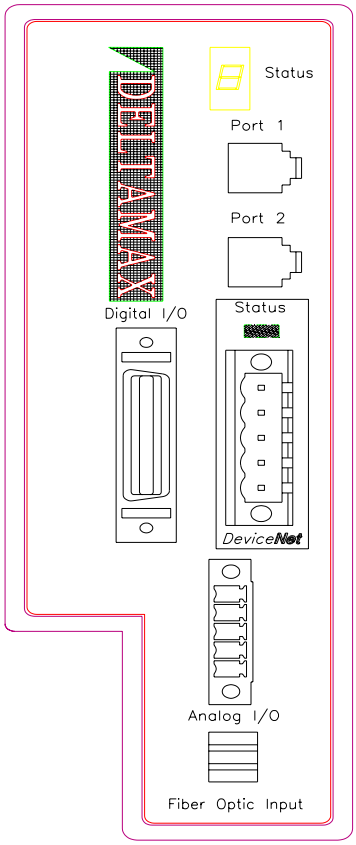
Device Profile:	Generic
Product Name:	DeltaMax
Product Code:	0
Produuct Revision:	1

DeltaMax Object Model

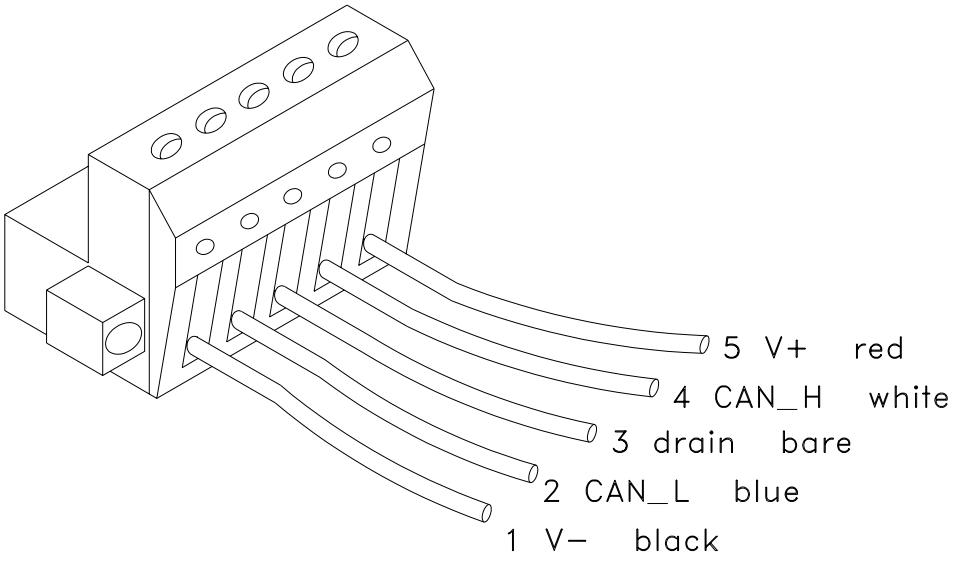


Connecting DeltaMax to DeviceNet Network

The figure below shows the location of the DeviceNet connector on the DeltaMax.



The figure below shows the open pluggable terminal connector required for the DeviceNet Network connections.



Deltamax Objects

Flags

DeviceNet has limited access to DeltaMax FLAG bit map memory. FLAG memory in a DeltaMax contains vital information for the programmer developing a MacroProgram for any application. This bitmap is broken down as follows:

FLAG #	DESCRIPTION
0-15	Discrete Inputs; Application specific, user programmable, originate from an external connector.
16-23	Discrete Outputs; Application specific, user programmable, output to an external connector.
24-31	Software programmable limit switches.
32-55	DeviceNet input flags. The **DeviceNet** port may **read/write** to these flags. To be considered as inputs only by a MacroProgrammer, therefore not write-able by a MacroProgram.
56-71	DeviceNet output flags. The **DeviceNet** port may only **read** these flags. To be considered as outputs from the MacroProgram, they may be written too or read from in a MacroProgram.
72-79	Timers
80-95	Axis1 status flags; example JOGGING,BUSY,DOWN,INDEXING, ect..
96-111	Pseudo Axis status flags, similar to Axis1 status flags, however this is a software axis
112-127	Analog status flags.
128-132	DeviceNet Status flags; status flags read only. The **DeviceNet** port may only **read** these flags.
133-207	Not Used
208-255	User flags; have programmable effects in motion program.

Flag Object Definition

Class Code: 100 (64HEX)

Instances: 1-256, one instance for each flag. Flags are zero based therefore instance 1 is for flag zero.

Instance Attributes: (one only)

Attr. ID	Access Rule	Attr. Name	Data Type	Description
3	get/set	Value	BOOL	The logic value of the flag

The access rule for each instance is dependant on the flag# associated with it as follows:

Instance	Flag#	Access Rule
1-32	0-31	none
33-56	32-55	get/set
57-72	56-71	get
73-128	72-127	none
129-144	128-143	get
145-256	144-255	none

Instance Definitions

33 thru 56) These instances are DeviceNet input flags, therefore they are read-only MacroProgram flags. A device external to the DeltaMax may set and clear these flags over DeviceNet.

57 thru 72) These instances are DeviceNet output flags, therefore they may be set or cleared in a MacroProgram. A device external to DeltaMax may access their attributes over DeviceNet.

129) CanBus Enabled Status Flag. This instance is TRUE once a canset instruction is succesfully executed in a MacroProgram.

130) CanBus Fault Flag. This instance is TRUE if DeviceNet interface goes busOff.

131) Explicit Connection Exists. This instance is set while an explicit connection is present.
132) Polled Connection Exists. This instance is set while a polled connection is present.

133) Bit-Strobed Connection Exists. This instance is set while a bit-strobed connection is present.

134) DeviceNet Bus Power Down. This instance is set when the DeltaMax hardware detects that the DeviceNet Network power has faulted. The flag is intended for MacroProgram use.

Data Memory

This object in the DeltaMax controller allows access to all of the MacroProgram variable memory area. It offers addressing modes that the individual Float and Integer Objects do not. It is intended to facilitate large block transfers of memory.

MacroProgram variable memory is set aside exclusively for an individual application. In total it is 16000 bytes, however it is broken into two fairly unique sections. One section holds data that is referred to in a MacroProgram of type integer, they are 32bits wide. The other section holds type floats, they are 64bits wide. The two sections have beginning address offsets of 0 and an ending offset depending on the application. The integer and float sections of data memory are contiguous, therefore the float address offset of 0 is actually address following the integer ending offset. Variable address offsets are easily obtained by the programmer in a symbol table file generated by the "MacroPro II" tool kit.

Data Memory Object Definition

Class Code: 103 (67HEX)

Instances: 1- 2 (1-Integers, 2-Floats)

Instance Attributes:

Attr. ID	Access Rule	Attr. Name	Data Type	Description
1	get/set	Address Offset	UINT	Address offset used for Attr. 3 when accessing the value. Default = 0.
2	get/set	Data Length	UCHAR	Data access length 1..8 Defaults: 4 for integers instance 1 8 for floats instance 2
3	get/set	Value	DINT LINT	The data memory value. The data type can be modified with attribute 2.
4	get/set	Address Mode	UCHAR	Available for selecting an addressing mode, see table 1.1 Default = 0, address is in service data.
5	get	Unprotected Lo	UINT	Data is accessible for write above this offset. Default = 0
6	get	Unprotected Hi	UINT	Data is accessible for write below this offset. Default = ending offset, application specific.

Table 1 - Addressing Modes (attribute 4)

Value	Description
0	Address offset is obtained from service data.
1	Address is obtained from the address offset attribute, the attribute is not incremented after the access.
2	Address is obtained from the address offset attribute, with auto-increment, in that the address offset is incremented after the access by the "Data Length". Once the address reaches the end offset a Vendor Specific error is returned with add code of 01 (address offset out of range).

Data Memory Behavior

Attributes 5 and 6 are defaulted such that over DeviceNet MacroProgram variables may not be written to. If it is required that variables in a MacroProgram be written to over the Network then a **dnet_range** instruction must be executed in the MacroProgram, see **dnet_range** section herein.

Integer

The Integer Object allows easy access to Integer variables for most DeviceNet devices.

This object in the DeltaMax controller allows access to only the integers of the MacroProgram variable memory area. There is one instance for each integer of the MacroProgram. Due to the nature of DeviceNet, one is able to only address up to the first 256 instances of the Integer Memory space, however if need be one can access all of integer space using the Data Memory Object.

Integer Variable Memory Object Definition

Class Code: 101 (65HEX)

Instances: 1 thru 256

Instance Attributes:

Attr. ID	Access Rule	Attr. Name	Data Type	Description
1	get	Address Offset	UINT	Address offset of the instance.
2	get	Data Length	UCHAR	Always 4 bytes.
3	get/set	Value	DINT	Value of Integer variable instance.
4	get	Unprotected Lo	UINT	Same as Data Memory Object.
5	get	Unprotected Hi	UINT	Same as Data Memory Object.

Integer Variable Behavior

Attributes 4 and 5 are defaulted such that over DeviceNet MacroProgram variables may not be written to. If it is required that variables in a MacroProgram be written to over the Network then a **dnet_range** instruction must be executed in the MacroProgram, see **dnet_range** section herein.

Float

The Integer Object allows easy access to Integer variables for most DeviceNet devices.

This object in the DeltaMax controller allows access to only the floats of the MacroProgram variable memory area. There is one instance for each float of the MacroProgram. Due to the nature of DeviceNet, one is able to only address up to the first 256 instances of Float Memory space, however if need be one can access all of the float space using the Data Memory Object.

Float Variable Memory Object Definition

Class Code: 102 (66HEX)

Instances: 1 thru 256

Instance Attributes:

Attr. ID	Access Rule	Attr. Name	Data Type	Description
1	get	Address Offset	UINT	Address offset of the instance.
2	get	Data Length	UCHAR	Always 8 bytes.
3	get/set	Value	LREAL	Value of Float variable instance.
4	get	Unprotected Lo	UINT	Same as Data Memory Object.
5	get	Unprotected Hi	UINT	Same as Data Memory Object.

Float Variable Behavior

Float variables are 64bits wide, they are defined in IEEE 754 for the basic doudle floating-point format.

Attributes 4 and 5 are defaulted such that over DeviceNet MacroProgram variables may not be written to. If it is required that variables in a MacroProgram be written to over the Network then a **dnet_range** instruction must be executed in the MacroProgram, see **dnet_range** section herein.

DeltaMax Programmable Parameter Object

The DeltaMax Parameter object is programmable via macro instructions. This allows for unique application specific requirements to be accessed over the DeviceNet network as a parameter. Tools like DeviceNet Manager by Allen-Bradley may then easily upload the parameter instances displaying their value and text description for user friendly interface to the controllers MacroProgram. DeviceNet Manager also has the capability to then generate a DeviceNet EDS File (electronic data sheet) from the information obtained from a Parameter Object Instance.

The Parameter Object Instance attributes are defined by ODVA as follows:

Attribute Name (ID)	[Expected Values, Responses] Full Parameter Instance
Undefined (00)	[Attribute_Not_Supported (x14)]
Parameter_Value (01)	Depends on Attributes 5, 6
Link_Path_Size (02)	[USINT (0..255)]
Link_Path (03)	[PATH_ID[Link_Path_Size] see Note 1 below]
Descriptor (04)	[WORD (0000000000xxxxxx)]
Data_Type (05)	[USINT (1..26)]
Data_Size (06)	[USINT (1..255)]
Parameter_Name_String (07)	[SHORT_STRING with max of 16 characters]
Units_String (08)	[SHORT_STRING with max of 4 characters]
Help_String (09)	[SHORT_STRING with max of 64 characters]
Minimum_Value (10)	[Same type as Parameter_Value (01)]
Maximum_Value (11)	[Same type as Parameter_Value (01)]
Default_Value (12)	[Same type as Parameter_Value (01)]
Scaling_Multiplier (13)	[UINT (1..65535)]
Scaling_Divisor (14)	[UINT (1..65535)]
Scaling_Base (15)	[UINT (0..65535)]
Scaling_Offset (16)	[INT (-32768..32767)]
Multiplier_Link (17)	[UINT (1..65535)]
Divisor_Link (18)	[UINT (1..65535)]
Base_Link (19)	[UINT (0..65535)]
Offset_Link (20)	[UINT (0..65535)]
Decimal_Precision (21)	[USINT (0..255)]
Undefined (22..99)	[Attribute_Not_Supported]

Parameters may be programmed for a Flag, Integer, or Float Object instances. Instances of the Flag, Interger and Float Objects are just varaibles declared in a MacroProgram. Once the variables are declared in a motion application program then the following MacroProgram instructions may be added to the program to generate a parameter object for DeviceNet Manager Tools:

dnet_flag_pm
dnet_int_pm
dnet_flt_pm

The instructions to program a parameter object must be contained within a **begin_cfg** and **end_cfg** instruction block, also a **begin_cfg** instruction in a MacroProgram must be the first executable instruction in a program. The usage of the above instructions is defined in the MacroProgram Features section of this document, also refer to the example programs.

The **dnet_range** instruction must be executed prior to a **dnet_int_pm** or **dnet_flt_pm** for any parameters to be any thing but **READ ONLY**. This also effects how parameters behave within any DeviceNet managing tools.

It is recommended that a parameter object be programmed into a DeltaMax prior to enabling the DeviceNet port. the DeviceNet port is enabled once a **can_set** instruction is executed.

I/O Connections

The DeltaMax has two possible static I/O instances of the Connection Object, see appendix A, one Polled (Instance 2) and one Bit-Strobed (Instance 3).

Polled Connection

The DeltaMax will consume 3 bytes in a polled command. These three bytes will set or clear flag numbers 32 thru 55, instances 33 thru 56 resp., of the Flags Object as follows.

	Bit Numbers			Bit Numbers			Bit Numbers		
	7	BYTE 1	0	7	BYTE 2	0	7	BYTE 3	0
flag#	39		32	47		40	55		48

The DeltaMax will produce 2 bytes in the polled response, the bytes will contain the present condition of flag numbers 56 thru 71, instances 57 thru 72, of the Flags Object as follows.

	Bit Numbers			Bit Numbers		
	7	BYTE 1	0	7	BYTE 2	0
flag#	63		56	71		64

Bit Strobed Connection

The DeltaMax shall produce only from a Bit-Strobed command. The response shall contain the present state of flag numbers 56 thru 71. The Data bytes in the response are indentical to the Polled connection response as follows.

	Bit Numbers			Bit Numbers		
	7	BYTE 1	0	7	BYTE 2	0
flag#	63		56	71		64

Important Note: A Bit-Strobed connection when allocated requires the DeltaMax processor to listen to all Group 2 DeveNet traffic on the bus. Under normal bus conditions this should not be a significant borden, however it could possibly slow MacroProgram processing under very heavy DeviceNet traffic.

MACROPRO II™

FOR WINDOWS™

Section 5 - Analyzer

INDUSTRIAL INDEXING SYSTEMS, INC.

SECTION 5 - ANALYZER

Table of Contents

Analyzer Help

The MacroPro II Analyzer has many different features that allow you to analyze the operation of an Industrial Indexing Systems DeltaMax.

Parameters Difference Message

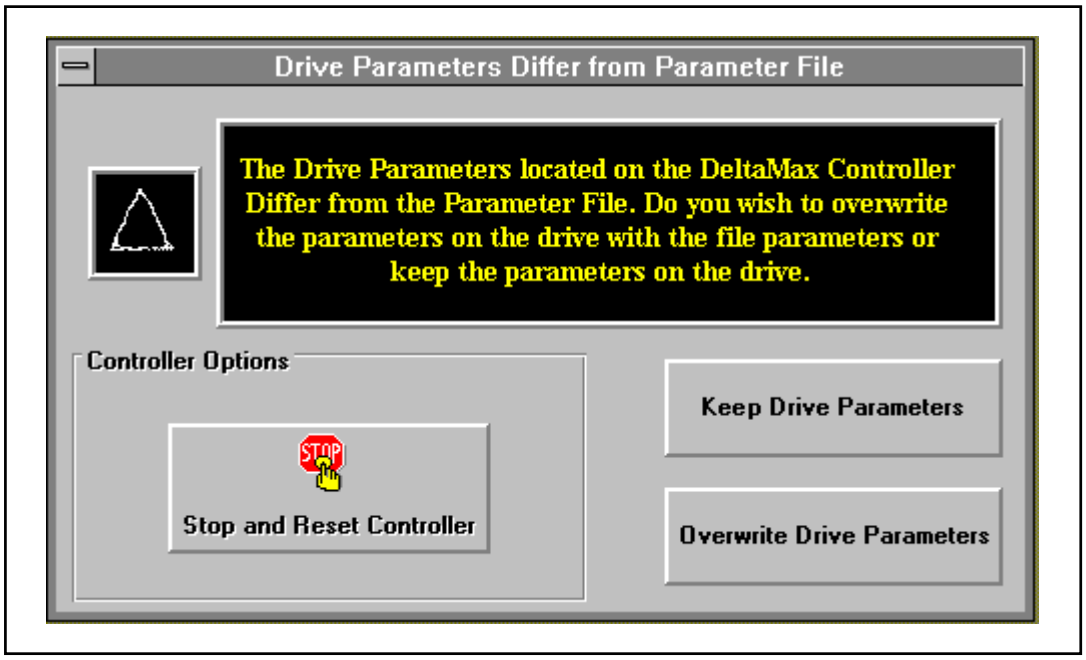


Figure 4.1 - Parameters Difference Display

Parameters Difference Display

This area displays the status of the ".par" (user parameters) file associated with the program in the PC as compared with the ".par" file currently stored in the drive.

Keep Drive Parameters

Selecting this button will load the parameters from the drive into the ".par" file associated with the program currently in the PC.

Overwrite Drive Parameters

This button will write the ".par" file associated with this program into the DeltaMax. This button will only be active if the program in the controller is stopped and reset. The controller needs to be loaded and started after this command is executed.

Stop and Reset Controller

This button is only active if the controller is not reset. Stop and Reset will cease the program running in the DeltaMax, if one is running, and reset the DeltaMax. An **f_decel** (forced deceleration) command is sent to the axis controller and the pseudo controller.

NOTE: When a Stop and Reset Controller is issued, the DeltaMax program memory is cleared, all flags are cleared, and any existing program is erased. Any Input modules treated as Output modules using MacroPro II will be reset as Inputs. Any Output modules treated as PLS modules in the program will be reset as Outputs.

Analyzer Tool Bar

Figure 4.2 - Analyzer Menu, Tool, and Status Bars

Message Bar

This line is where the Analyzer notifies the user about the status of a current operation.

Program in PC

This is the name of the program that is currently in the PC.

Equality Indicator

This indicator will either be an equals sign or an unequal sign. An equals sign indicates that the program in the PC is exactly the same as the program in the DeltaMax, whereas an unequal sign indicates that there is a difference between the two programs in the PC and the DeltaMax (date and time stamps are included in the comparison).

Program in DeltaMax

This is the name of the program that is currently loaded in the DeltaMax.

File

To access the File feature, use the [Alt + f] key combination or select File from the Menu Item Bar.

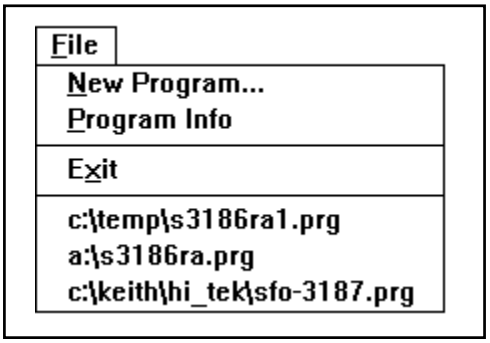


Figure 4.3 - Analyzer File Menu

New Program

New Program will display the standard file open dialog. The default display will show all of the ".prg" source files in the selected directory. The only files that can be loaded into the Analyzer are the ".prg" files. The other associated files will be loaded automatically as required.

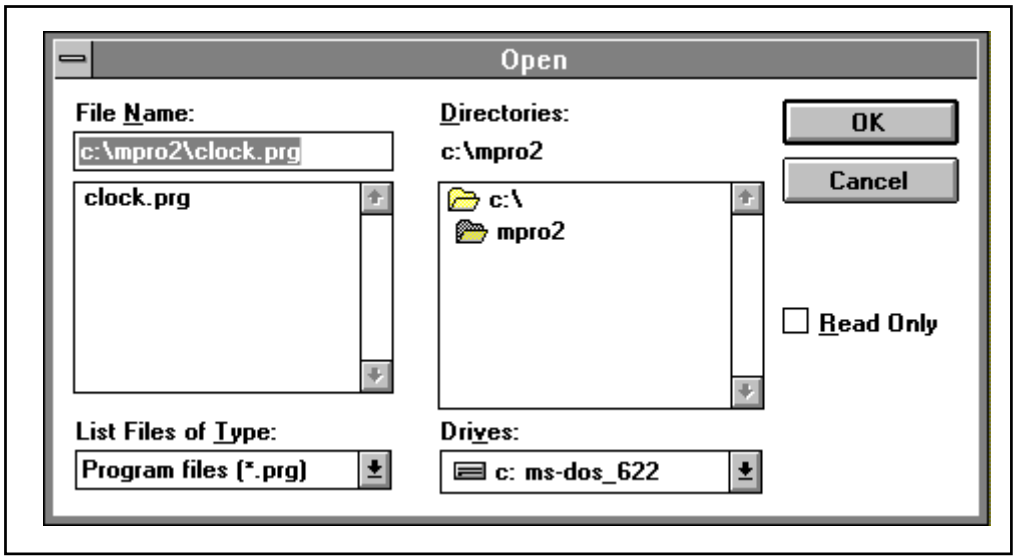


Figure 4.4 - New Program Sample Screen

NOTE: There are a number of associated files that must be present to allow full functionality of the Analyzer. These are: ".mcx" (the file that is loaded into the DeltaMax), ".dbg" (Analyzer information file), ".par" (user parameters), and ".sym" (symbol file, equates, labels, data, & constants).

Program Info

Program Info will display file information about the “Source” and “Executable” file that is resident in your PC. This information includes the file name, size, date and time. It is important to understand that the program loaded into your PC may or may not be the same as the program loaded into the DeltaMax.

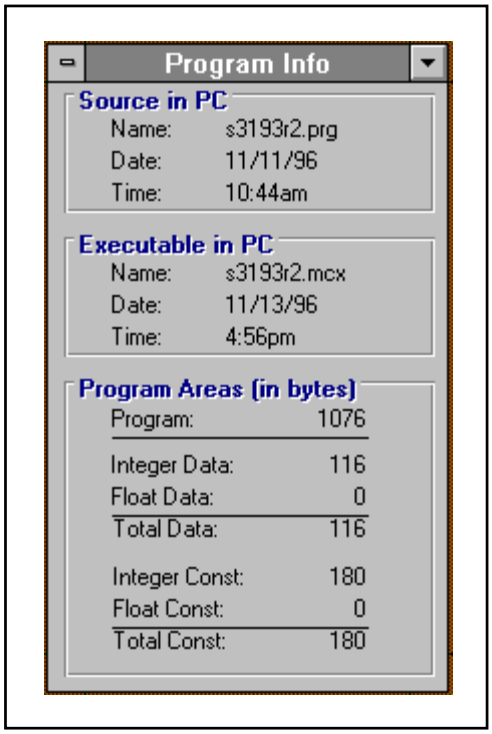


Figure 4.5 - Program Info Sample Screen

NOTE: To view information about the program loaded into your DeltaMax, use the DeltaMax Info function.

Exit

Exit Analyzer will close all open Analyzer windows.

Previous Programs

This section will allow the user to select a previously used program.

Actions

To access the Actions feature, use the [Alt +a] combination or select Actions from the Menu Item Bar.

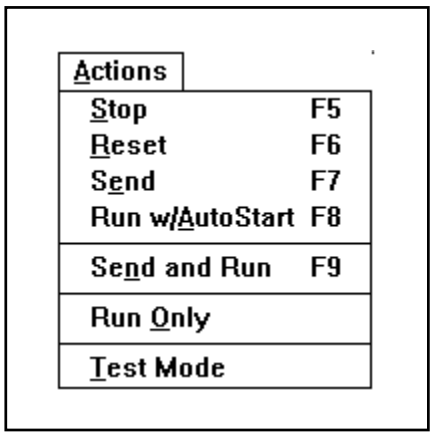


Figure 4.6 - Analyzer Actions Menu

Stop

Stop will cease the program running in the DeltaMax, if one is running. An **f_decel** (forced deceleration) command is sent to the axis controller and the pseudo controller. The F5 function key will also execute this operation.

Reset

Reset will cause a reset of the DeltaMax. The current program must be stopped before a Reset is allowed. The F6 function key will also execute this operation.

NOTE: When a Reset is issued, the DeltaMax program memory is cleared, all flags are cleared, and any existing program is erased. Any Input modules treated as Output modules using MacroPro II will be reset as Inputs. Any Output modules treated as PLS modules in the program will be reset as Outputs.

Send

Send will download the current program executable file (.mcx) from the PC to the DeltaMax. A Reset must be performed prior to using the Send function. The F7 function key will also execute this operation.

Run w/AutoStart

Run w/AutoStart will turn on the AutoStart bit in the DeltaMax's status word. The DeltaMax will test its AutoStart bit on startup and, if set, the loaded program will start automatically. If the program is stopped, setting AutoStart will start the program in the DeltaMax in addition to setting the AutoStart bit. The F8 function key will execute this operation. The only way to clear the AutoStart is to issue a Reset.

Send and Run

Send and Run will issue the Stop, Reset, Send, and Run w/AutoStart commands to the DeltaMax. The F9 function key will execute this operation.

Run Only

Run will start or re-start the program currently resident in the DeltaMax. If a program is not loaded, this function has no effect on the DeltaMax.

Test Mode

This function has not been implemented at the time this manual was written.

Status

To access the Status feature, use the [Alt + s] key combination or select Status from the Menu Item Bar.

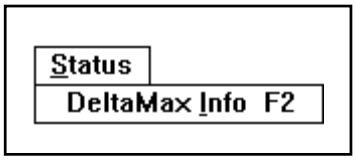


Figure 4.7 - Analyzer Status Menu

DeltaMax Info

DeltaMax Info will display a window showing Program Information, the DeltaMax System Status, the part number of the DeltaMax Controller, and the Status Information for the selected controller. Pressing F2 will also execute this operation.

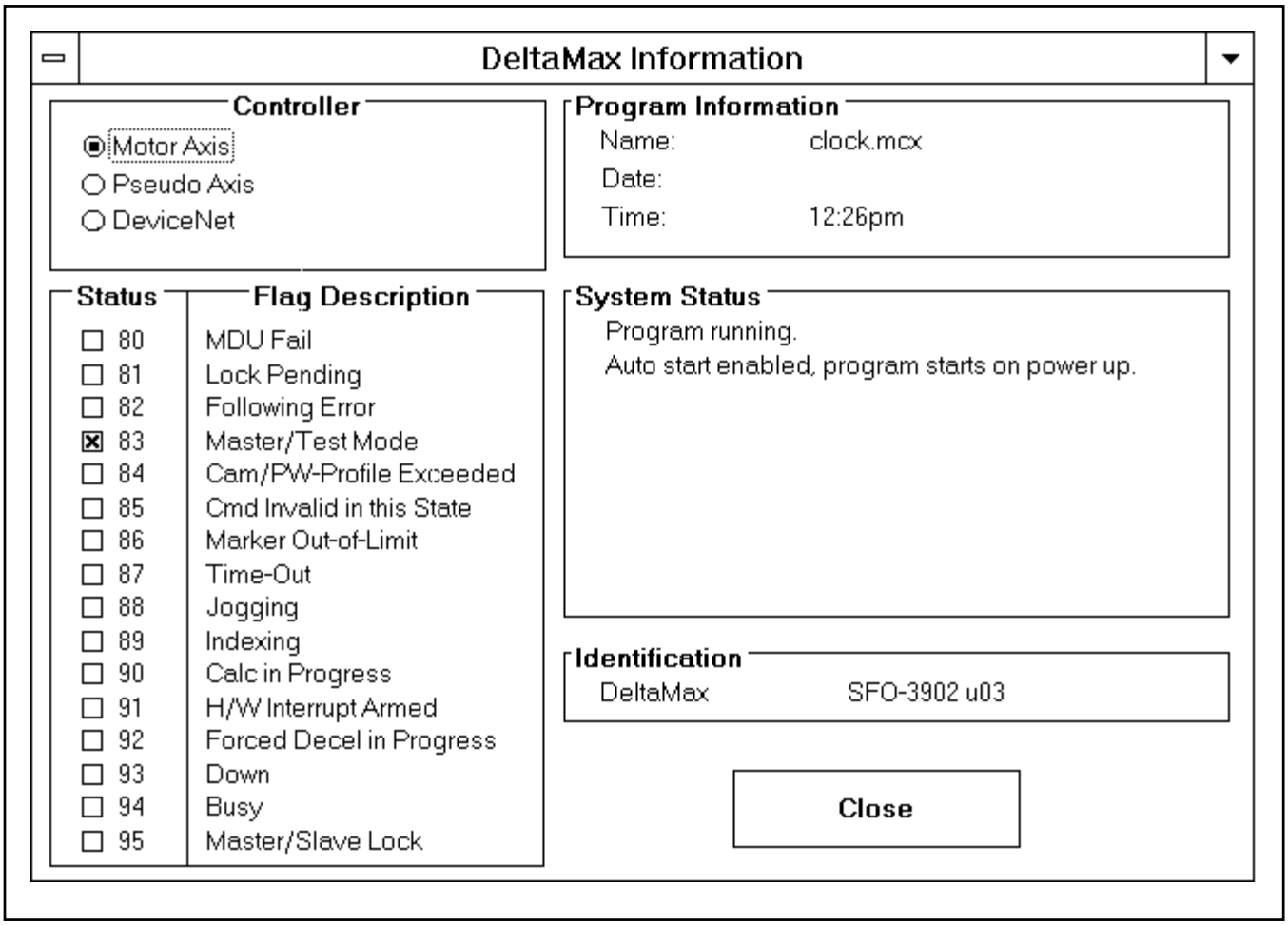


Figure 4.8 - DeltaMax Info Sample Screen

Controller Selection

DeltaMax Controller Status will display a window showing the status information for a single controller. This includes the flag number of the 16 associated status flags for that controller as well as a description of each flag. Active Status flags are indicated as such by a check in the corresponding check box.

DeltaMax Program Information

This section of the window indicates the name of the program resident on the DeltaMax along with the date and time that this program was compiled or created.

Status Indicators

The left section of the window displays the number of the Status Flag along with a check box which indicates whether or not the listed flag is active.

Flag Description

The right section of the window lists the Description of each Status Flag. The Flag Descriptions will change depending on the type of controller selected.

DeltaMax System Status

This section of the window lists all System Status messages, such as “Program stopped” or “Program running”.

DeltaMax Identification

This section of the window indicates the part number and revision of the firmware (operating system) used. The firmware part numbers are shown as SFONNNNRXX . “SFO” stands for System Functional Operation, “NNNN” is a number from 0000 to 9999 and “XX” is a revision ranging from 00 to 99.

Close

Clicking on this selection will close the DeltaMax Information window.

Parameters Menu Item

To access the Status feature, use the [Alt + p] key combination or select Parameters from the Menu Item Bar.

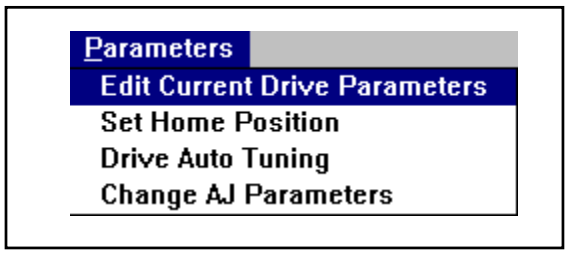


Figure 4.9 - Analyzer Parameters Menu

Parameters

Parameters will display a window showing the status of the current drive parameters and allow these values to be edited.

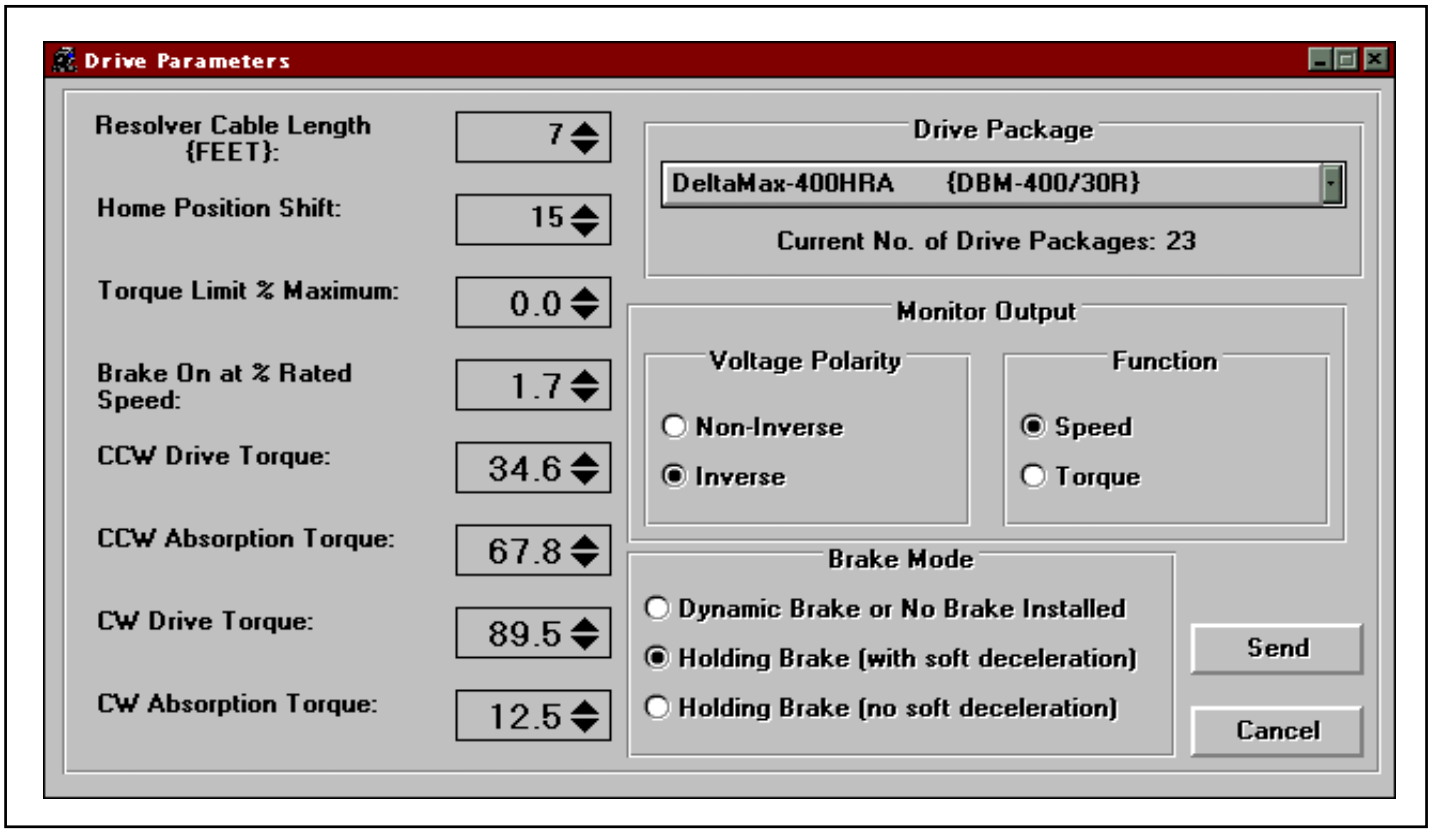


Figure 4.10 - Parameters Sample Screen

Resolver Cable Length

Sets the driver resolver cable length compensation. The driver power must be cycled to register this value. The range of acceptable values is from 3 to 394 feet.

Home Position Shift

The resolver zero can be shifted from the mechanical motor shaft position using Home Position Shift. This position can be shifted 2047 bits.

Torque Limit % Maximum

When the torque limit is applied the motor torque is limited by the Torque Limit % Maximum. This value is the % of motor peak torque and can range from 0 to 100%. When this value is set to 0 the selections for CCW Drive Torque, CW Drive Torque, CCW Absorption Torque and CW Absorption Torque become available.

Brake On at % Rated Speed

When Brake Mode is set up for Holding Brake (with soft deceleration), the mechanical holding brake is applied when the motor speed falls below this percentage. The deceleration is controlled at the set deceleration rate. Brake On at % Rated Speed is scaled in % of rated speed.

CCW Drive Torque

This value limits the CCW rotation torque. The range is 0 to 100% with 100% being the Peak Torque. This parameter becomes available when the Torque Limit % Maximum is set to 0.

CCW Absorption Torque

This value limits the CCW rotation absorption (braking) torque. The range is 0 to 100% with 100% being the Peak Torque. This parameter becomes available when the Torque Limit % Maximum is set to 0.

Drive_Package

The Drive Package parameter sets the internal driver parameters corresponding to the Drive Package. The driver power must be cycled to register this parameter.

Monitor Output

The Monitor Output selects the type of signal that is sent to the external MON output pins on the driver. The signal can either be speed or torque and normal or inverted.

Brake Mode

Selects the type of brake sequencing to be done when the drive is turned off. Dynamic Brake is used when no brake is installed. After the drive is turned off The Holding Brake (with soft deceleration) is applied after controlled deceleration to the speed set by the Brake On at % Rated Speed and is applied immediately in the case of a fault. Holding Brake (no soft deceleration) is applied immediately after the drive is turned off.

Send Drive Parameters

Send will load the currently displayed parameters into the controller. The program must be stopped and reset before the new parameters can be written to the drive.

Cancel Drive Parameters

The Cancel button will exit the Drive Parameters menu without sending the drive information to the controller.

Home Position

Click on an item below for help on that particular selection:

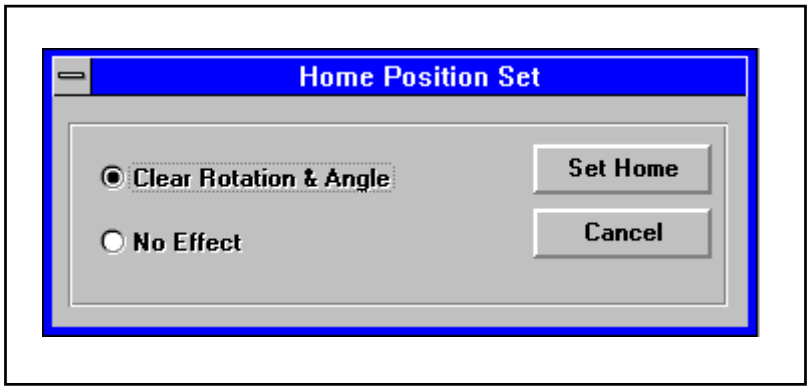


Figure 4.11 - Home Position Set Screen

Home Position Description

Home Position Set will display a window that will allow the user to set the absolute home position any time. The Tool Bar button will execute this operation.

Home Position Set

Sets the absolute zero position, clears all the turns and angle. The '**Clear Rotation & Angle**' needs to be selected and the '**Set Home**' needs to be pushed for this to take effect. If '**No Effect**' is chosen nothing will happen. ' **Cancel**' will cancel the operation.

Auto Tune

Click on an item below for help on that particular selection.
Note: The user can start auto tuning after clearing the warning.

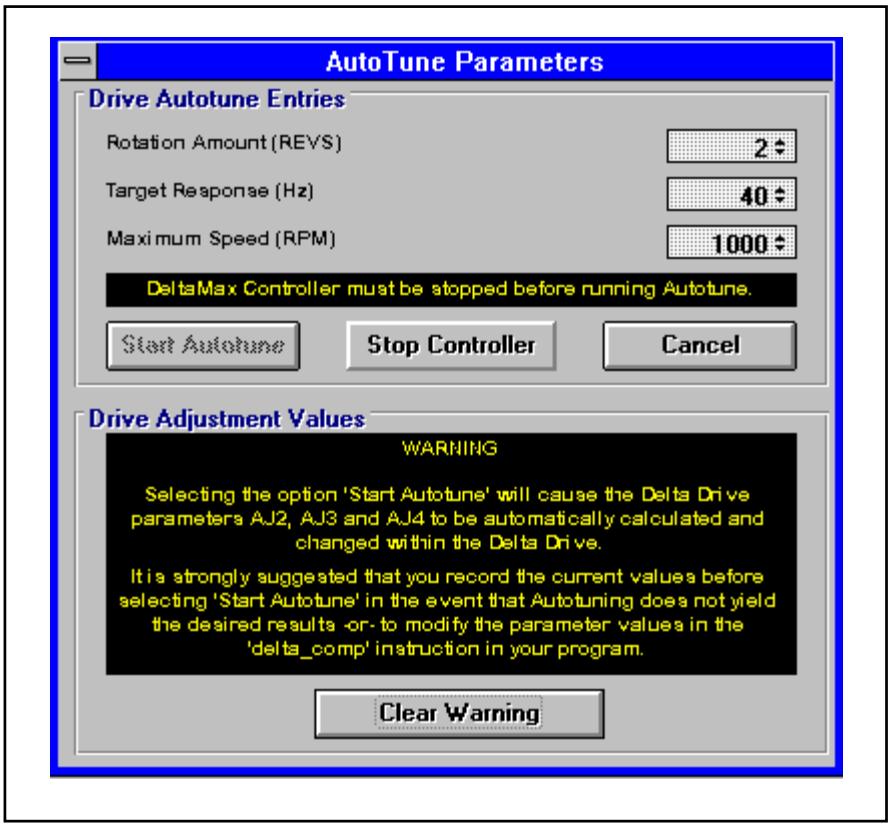


Figure 4.12 - Drive Auto Tune Screen

Drive Auto Tune

The following display allows the user to auto tune the drive by entering the desired values. At the end of the auto tuning sequence, AJ2, AJ3 and AJ4 will be set automatically and the values will be displayed under '*Drive Adjustment Values*'.

Click on an item below for help on that particular selection.

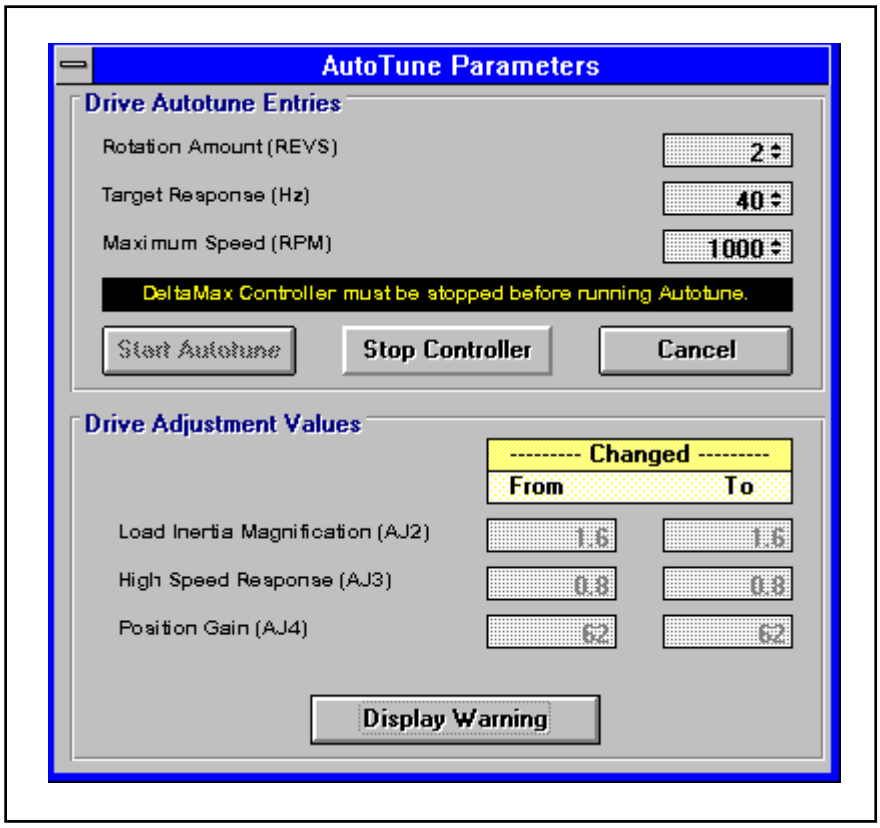


Figure 4.12a - Drive Auto Tune Screen

Rotation Amount

Sets the amount of the reciprocal rotation during the auto tuning sequence.
Valid Range: 1 to 300 revs

Target Response

Sets the desired frequency response. If this value is too high, unstable operation may result.
Valid Range: 1 to 1000 Hz

Maximum Speed

Sets the speed of the reciprocal rotation during the auto tune sequence.
Valid Range: 1 to 4000 rpm

Start Autotune

This starts the auto tune sequence.

AJ Parameters

Click on an item below for help on that particular selection.
Note: The user can modify the AJ parameters after clearing the warning.

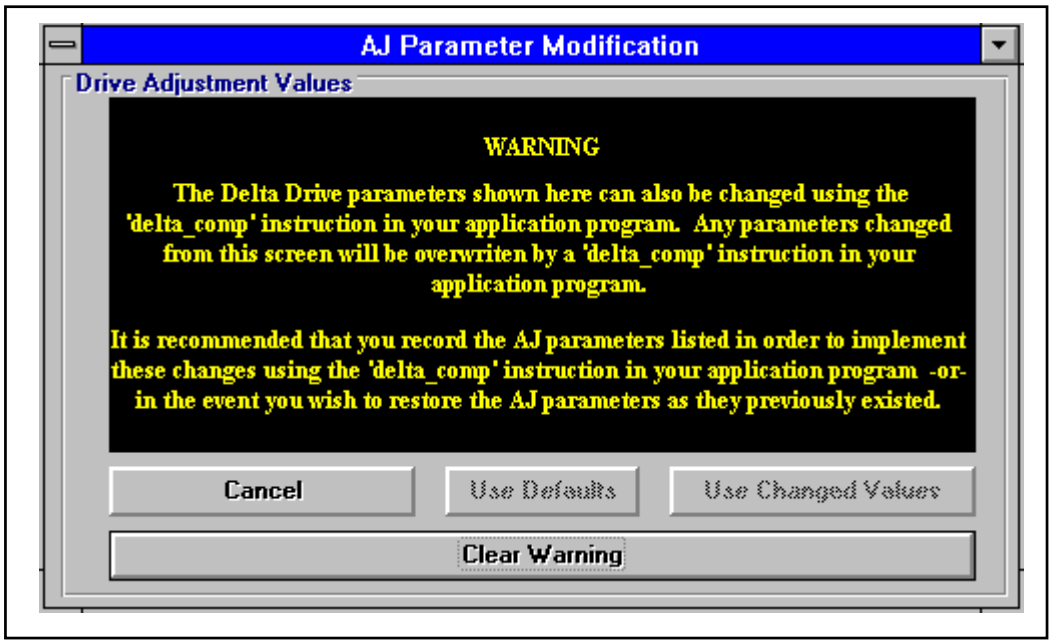


Figure 4.13 - AJ Parameters Screen

AJ Parameter Modification

This display allows the user to change the AJ parameters in the drive. Click on an item below for help on that particular selection.

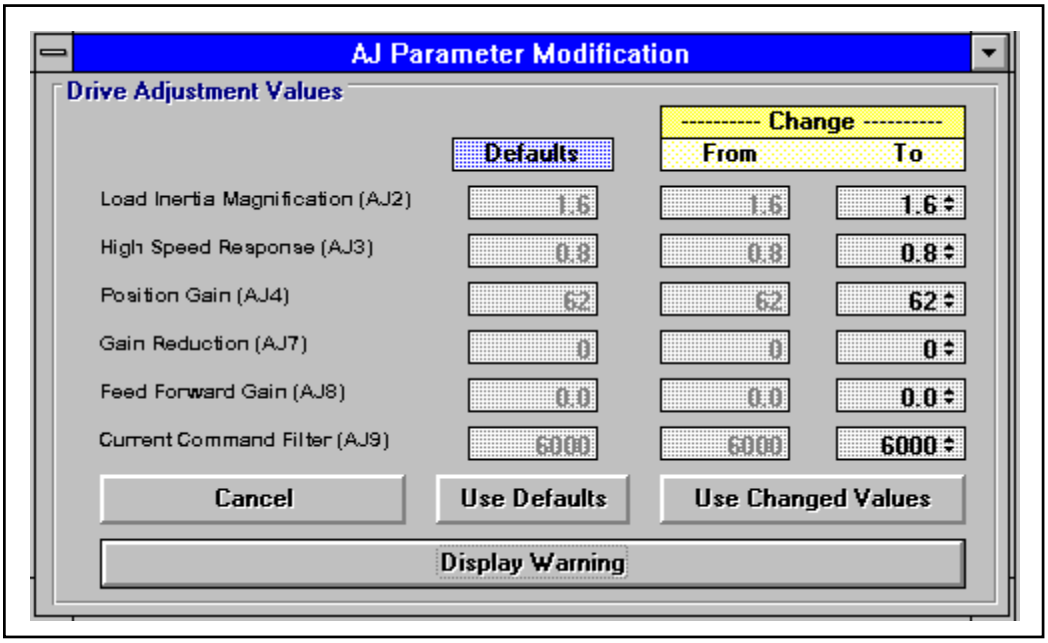


Figure 4.13a - AJ Parameters Screen

Load Inertia Magnification

Sets the baseline frequency response of the driver using the ratio of the load inertia/motor inertia for a rigidly coupled load. If the value is set too high, the motor and driver may become unstable and oscillate. This parameter is set automatically during auto tuning.

High Speed Response

Sets the high frequency response of the driver. The higher the number the more responsive. If the value is set too high, the motor and driver may become unstable and oscillate. This parameter is set automatically during auto tuning.

Position Gain

Sets the DC gain of the position control loop. The higher this value the more stiffer and faster the response. If the value is set too high, the motor and driver may become unstable and oscillate. This parameter is set automatically during auto tuning.

Gain Reduction

Sets the amount of gain reduction at zero speed.

Feed Forward Gain

Sets the feed forward gain in the position loop. A value of 1.0 results in 0.0 following error. Less than 1.0 will produce a lag between the actual motor position and the commanded position. A value greater than 1.0 produces a lead. The lead or lag will be proportional to speed for settings different from 1.0.

Current Command Filter

Sets the notch frequency of a velocity loop anti-resonance filter. This filter can be used to cancel machine or servo resonance.
Power must be turned OFF then ON for this parameter to take effect.

Change Values

Changes the AJ parameters to the values set by the user.

Cancel

The Cancel button will end the current operation.

Read/Write

To access the File feature, use the [Alt + r] key combination or select File from the Menu Item Bar.

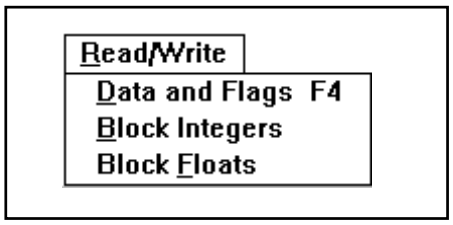


Figure 4.14 - Analyzer Read/Write Menu

Data And Flags

This window is used for reading individual data and/or flags from the DeltaMax and also for writing data and/or flags to the DeltaMax. Pressing F4 will also execute this operation.

NOTE: When reading cam information(arrays), displaying in hexadecimal format is recommended, which can be accomplished in the Block Integer section.

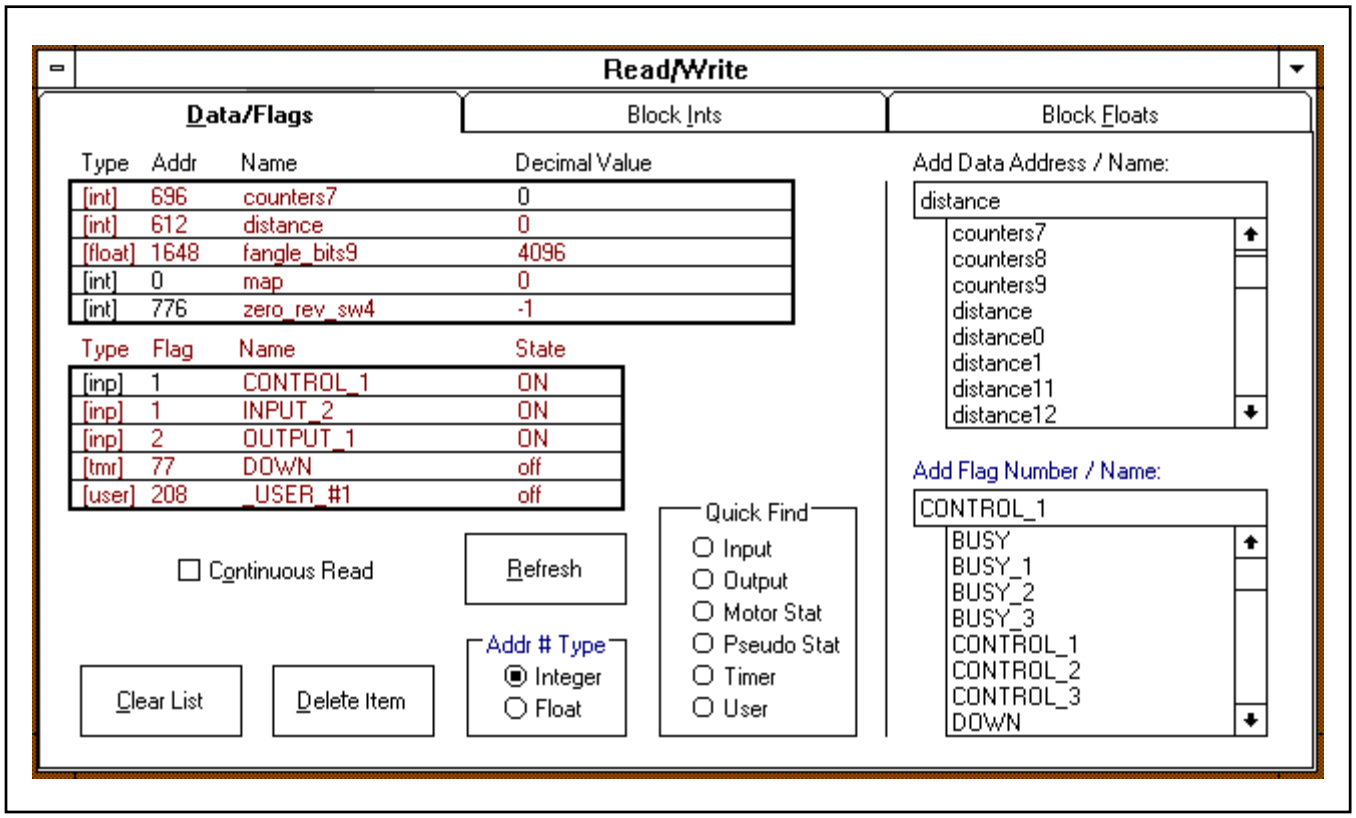


Figure 4.15 - Read/Write Sample Screen (Data and Flags)

Writing Data

To write data, click on the value of the data to write. Type in the value, in decimal only, and hit the 'Enter' key to write the value. By simply typing in a new value and hitting the 'Enter' key, another write will be performed.

Writing Flags

To change the state of a flag, double-click on the flag state. This will toggle the state of the flag. Timers and Status flags are read-only.

NOTE: Changing the state of an I/O flag from this window assigns that flag as an output flag and will no longer respond to an external input.

Continuous Read

Checking this item will start reading the selected data and/or flags from the DeltaMax and the headings are colored blue and names, values, etc. are black. When this item is not checked, the "Refresh" button appears and the continuous reading effect is turned off. Also, the headings and the values are colored red. The state of this button does not effect data/flag writes, they operate as normal when continuous read is on or off.

NOTE: If "Data and Flags", "Block Integer", or "Block Float" are set for reading, they only read when they are the active tab.

Refresh Button

Clicking this button will force a read of the read/write list. This button is visible only when the Continuous Read check box is not checked.

Address Number Type

When adding an address number to the read/write list, this specifies which data type it represents.

Add Data Name/Address

Type in (or select from the list) a data name or address to add to the read/write list. If it is an array name, put the element number in square brackets. If it is an address number, the address must be greater than or equal to zero and also divisible by 4 for integers (8 for floating point addresses). For the Data/Flags tab, the address number type can be specified by clicking on the appropriate option button in the Addr # Type frame.

Add Flag Name/Number

Type in (or select from the list) a flag name or flag number to add to the read/write list. If it is a flag number, it must be between 0 and 255. Flag names in the list starting with an underscore ("_") prefix are the predefined flags.

Block Integer

This window is used for reading blocks of integer data from the DeltaMax and also for writing integer data to the DeltaMax.

NOTE: When reading cam information(arrays), displaying in hexadecimal format is recommended.

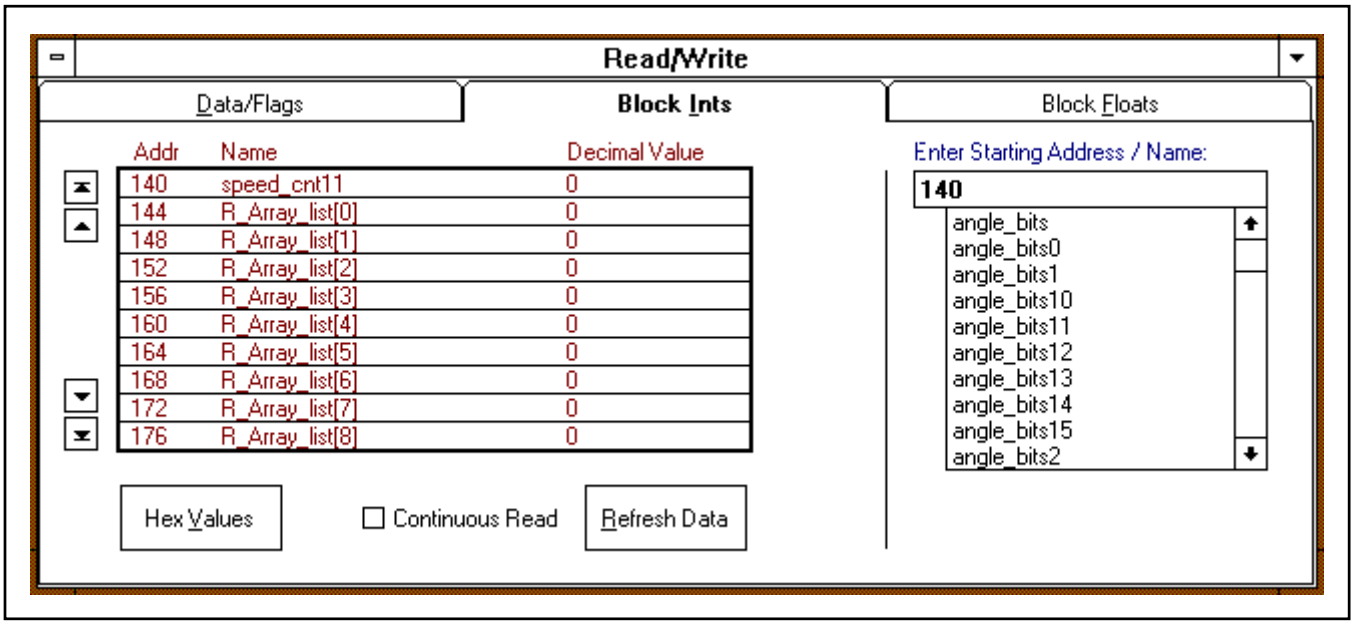


Figure 4.16 - Read/Write Sample Screen (Block Integer)

Block Float

This window is used for reading blocks of floating point data from the DeltaMax and also for writing floating point data to the DeltaMax. The Tool Bar button will execute this operation.

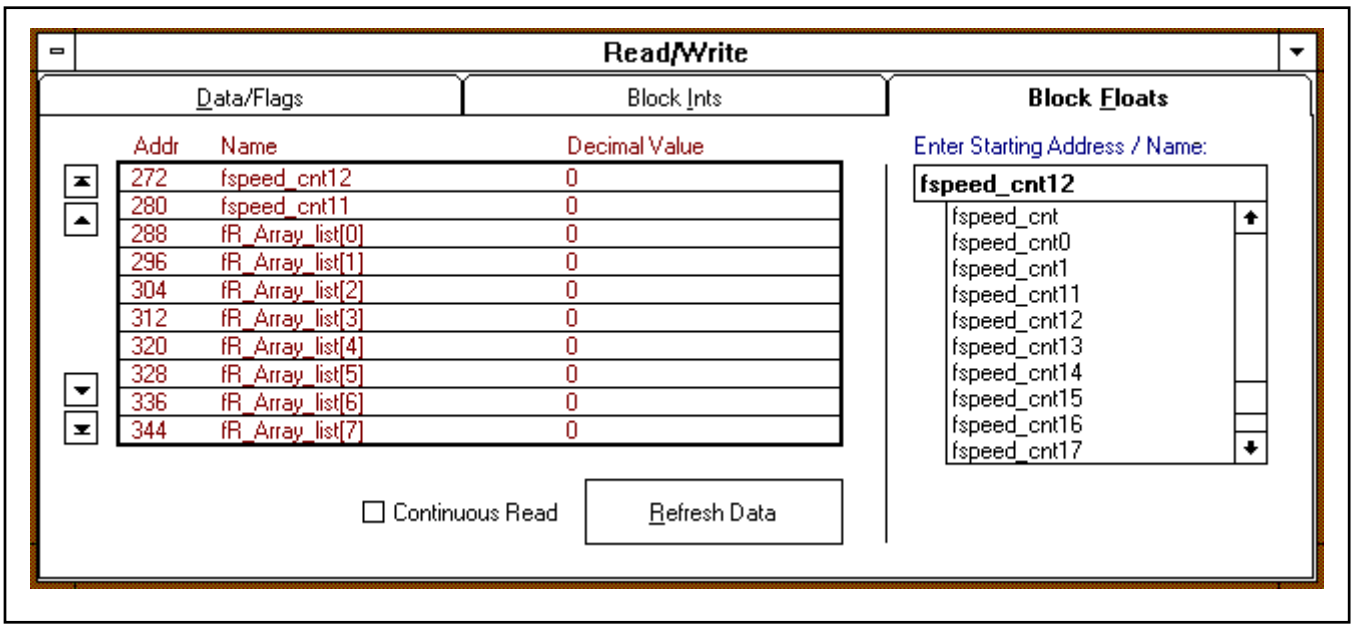


Figure 4.17 - Read/Write Sample Screen (Block Float)

Line Up/Down Button

Clicking this button increments or decrements the starting address by one data address.

Page Up/Down Button

Clicking this button increments or decrements the starting address by one block of 10 data addresses.

Decimal/Hex/ASCII Values Button

Toggling this button displays data in either decimal, hexadecimal, or ASCII format.

Starting Address Button

Click this button to change the starting address of the block of 10 data items. At the prompt enter a data label name or an address number. For array elements, enter the name followed by the element number enclosed in square brackets. For example, to enter a starting address of the 3rd element in the array named ***Cam***; simply enter ***Cam[2]*** (since arrays are zero based).

General Description

The Data & Flag List window is used to make up a list of data and flags to read/write. A maximum of 10 items can be used and may be any combination of data and/or flags. The Select Data list box is a list of variables defined in the current program (Source) and can be added to the read/write list.

Double-click on the data item to be added or select the item and then click the Add button.

Clear List Button

All items (data and/or flags) in the read/write list will be removed.

Delete Item

The selected data or flag will be removed from the read/write list. The selected item is that where the name, not value, is highlighted. Double-clicking the item will also delete it.

Read/Write List

Double-clicking an item will remove it from the list. Highlighting an item here activates the delete item button.

Use Button

Clicking this button will minimize the Data & Flags List window, show the Read/Write Data and Flags window and begin reading the items from the read/write list.

Quick Find Option Box

The Quick Find Option box will allow you to quickly select a predefined group of flags. Flags range from #0 to #255.

View

To access the File feature, use the [Alt + v] key combination or select File from the Menu Item Bar.

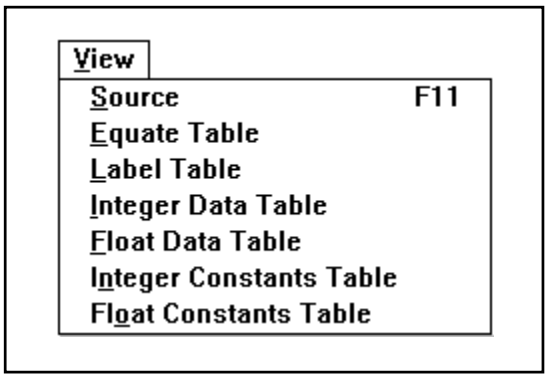


Figure 4.18 - Analyzer View Menu

Source

Source will display the Source for the program currently resident in the PC. Various controls allow you to view any or all program Source lines as needed. The F11 function key will also execute this operation.

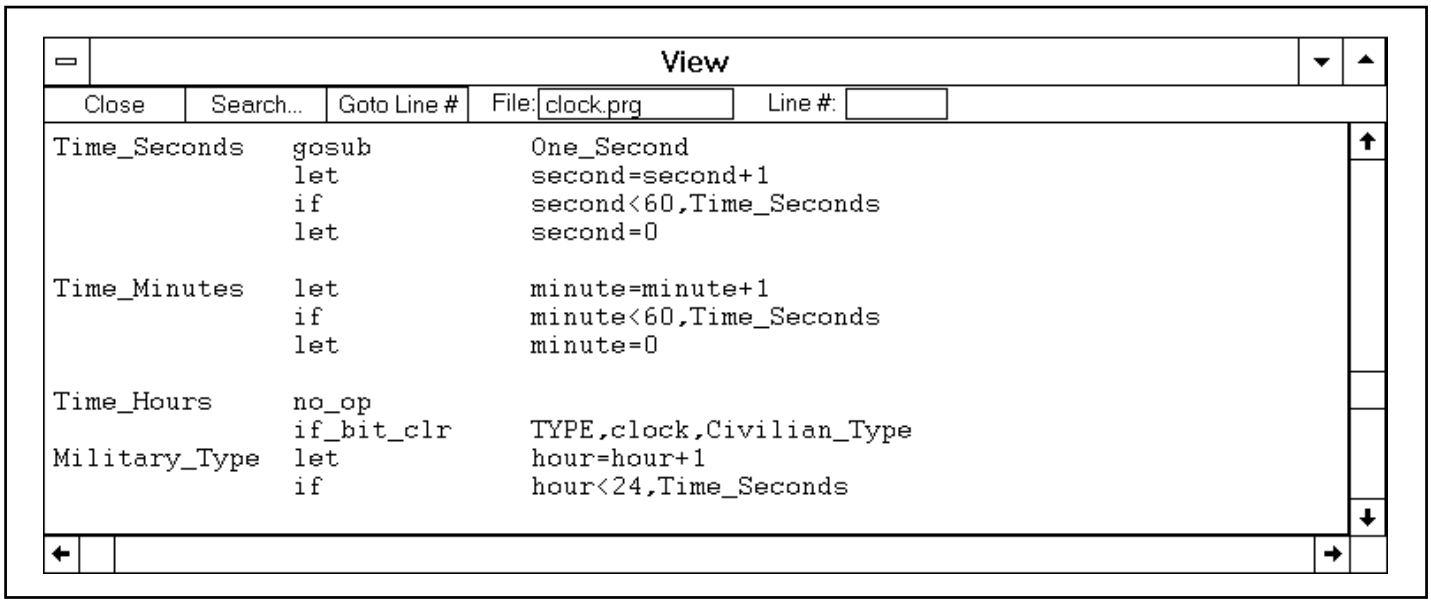


Figure 4.19 - View Sample Screen

Equate Table

Equate Table will move the file display pointer to the section in the symbol file (.sym) for the program currently resident in the PC, where all labels that were equated (using the **equ** instruction in the program) are listed. Various controls allow you to view any or all of the symbol file (.sym) as needed.

Label Table

Label Table will move the file display pointer to the section in the symbol file (.sym) for the program currently resident in the PC, where all program Labels are listed. Various controls allow you to view any or all of the symbol file (.sym) as needed.

Integer Constants Table

Integer Constants Table will move the file display pointer to the section in the symbol file (.sym) for the program currently resident in the PC, where all Integer Constants used in the program are listed. Various controls allow you to view any or all of the symbol file (.sym) as needed.

Float Constants Table

Float Constants Table will move the file display pointer to the section in the symbol file (.sym) for the program currently resident in the PC, where all Floating Point Constants used in the program are listed. Various controls allow you to view any or all of the symbol file (.sym) as needed.

Integer Data Table

Integer Data Table will move the file display pointer to the section in the symbol file (.sym) for the program currently resident in the PC, where all program Integer Data are listed. Various controls allow you to view any or all of the symbol file (.sym) as needed.

Float Data Table

Float Data Table will move the file display pointer to the section in the symbol file (.sym) for the program currently resident in the PC, where all program Floating Point Data are listed. Various controls allow you to view any or all of the symbol file (.sym) as needed.

Close

Close will exit the View window.

Search

Search allows you to find a designated text string within the file being viewed. A list of strings previously searched is maintained for easier viewing. Like most Search or find functions, the F3 key will search for the next occurrence of the entered string.

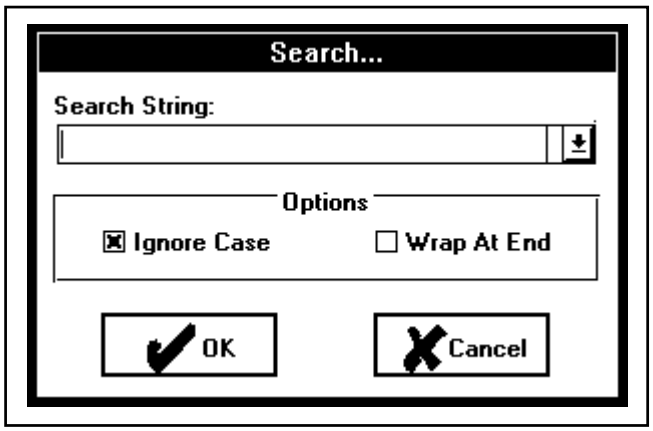


Figure 4.20 - Search Window

Search String

Enter the String to be searched, keeping in mind the Case option selected. A list of strings previously searched is maintained for easier viewing. Use the scroll bar to search on other strings as needed.

Ignore Case

When Ignore Case is checked, strings will be compared and the match will be made regardless of the case entered in the Search String *or* the file being searched.

Wrap at End

Wrap at End will cause the search to continue at the beginning of a file, when an end-of-file is encountered.

OK Button

Pressing the OK Button will cause the search for the selected/entered string to begin from the current file location.

Cancel Button

The Cancel Button will end the search and resume viewing the file.

Goto Line

Goto Line # lets you position to a specific line within the file being viewed. The range of allowable lines is listed.

Trace

The Trace feature allows you to follow the program execution without affecting the operation of the program.

To access the Trace feature, use the [Alt + t] key combination to access the Trace menu bar item.

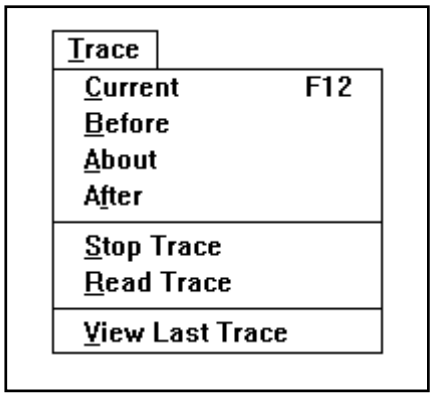


Figure 4.21 - Trace Menu Bar

Trace Window

The Trace feature is closely linked with the source file so that when the Trace window is opened, the Source view window is also opened. If the Source view window is already open, it may be re-sized so that the Trace and Source windows are equal in size.

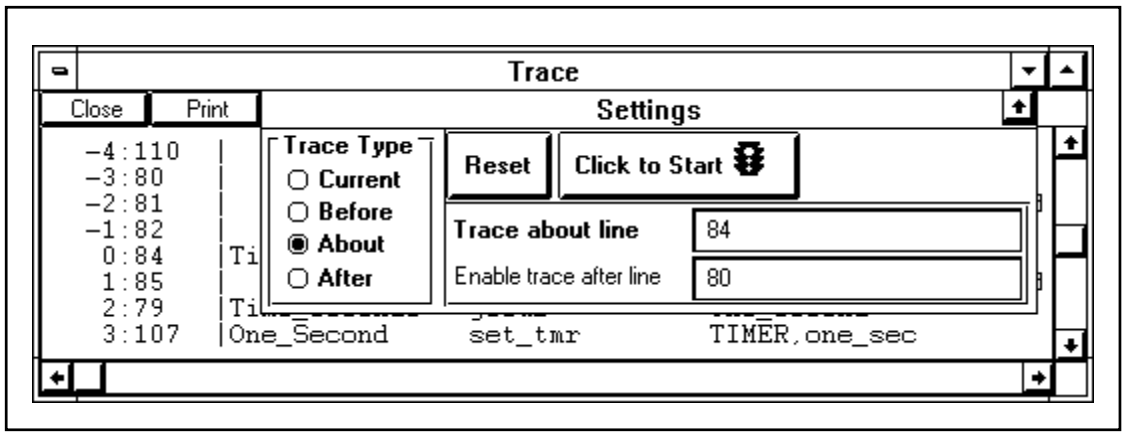


Figure 4.22 - Trace Window

The Trace Feature has the ability to capture up to 112 lines of program execution. This is not to say that each time the Trace is run that you should see 112 lines. The trace type, trace events, and program will effect the number of lines displayed after a successful trace.

After a successful trace, the “Settings” window will roll-up to show the full trace listing. The trace listing shows the Trc. # (trace number) in the first column. This trace number indicates the order in which a program line was executed. A negative number indicates lines executed before the trace line. The trace line numbered zero is the line that was specified as the line to trace on. Positive numbered lines are lines that were executed after the trace line was executed. The next column in the trace listing is the source program line number (Line #). The remaining columns are the actual source text for the associated line numbers.

There are two items in the menu bar Trace pull down box that are not available in the Trace window. These are Read Trace, and View Last Trace.

Helpful Hints

Clicking on a line in the Source window will automatically set the Trace Line or Enable Trace After Line. Repeated clicking on lines in the Source window will reset these trace events.

Highlighting a line in the Trace window will display the associated line in the Source window.

Entering any text string in the Trace Line or Enable Trace After Line entry boxes will bring up a Valid Label Selection window. This window will list all labels in the program that can be used as a trace parameter .

Current Trace Type

The Current trace type will initiate the Trace process regardless of the settings for Trace Line, and Enable Trace After Line. This trace type is useful when trying to determine if the program is running and what part of the program is being executed.

Before Trace Type

The Before trace type will allow you to see which lines are being executed before a particular program line. You must enter a Trace Line parameter to enable a trace of this type. The Enable Trace After Line is optional but will have an effect on the number of trace lines that might be displayed. The maximum number of lines that could be displayed before the “trace line” would be 112, with the trace line being Trc. # 0.

About Trace Type

The About trace type will allow you to see which lines are being executed before and after a specified program line. You must enter a Trace Line parameter to enable a trace of this type. The Enable Trace After Line is optional but will have an effect on the number of trace lines that might be displayed before the actual trace line. The maximum number of lines that could be displayed before the "trace line" would be 57 (negative Trc. #'s), with the trace line being Trc. # 0. The maximum number of trace lines that would be displayed after the trace line would be 55 (positive Trc. #'s).

After Trace Type

The After trace type will allow you to see which lines are being executed after a particular program line. You must enter a Trace Line parameter to enable a trace of this type. The Enable Trace After Line is optional. Your program will have an effect on the number of trace lines that might be displayed. The maximum number of lines that could be displayed after the "trace line" would be 112, with the trace line being Trc. # 0.

Stop Trace

This menu item will stop the currently active trace.

Close Command Button

The Close command button will stop any active trace events, then close the trace window.

Print Command Button

The Print command button will allow you to print the contents of the trace listing to your MS Windows print driver. This print uses the default settings that were last set with the Microsoft Windows Print Manager. It is recommended that you select a fixed space font (such as Courier), to give you the proper column alignment on the print out.

Reset Command Button

The Reset command button will stop any active trace and reset the Trace Line and Enable Trace After Line parameters.

Start/Stop Trace Command Button

The Start/Stop trace command button will initiate the trace process based on the settings listed in the "Settings" window, or stop an active trace. When a trace is active the "Trace Active" message panel will flash next to the Start/Stop trace command button, as shown the following figure.

Figure 4.23 - "Trace Active" Panel (next to Start/Stop Command Button)

Trace Line

Purpose:
The Trace Line entry specifies what is the focal point of the trace process. This entry identifies the line that triggers a trace in the DeltaMax. This parameter is required when using the Before, About and After trace types.

Entry method:
The Trace Line entry box accepts either a source line number or an Executable Source Line Labels. Checks are made on the validity of those entries. To enter a line number you can enter the number manually in the box or you can click on a line in the Source window. When you click on a line in the Source window the associated line number will be automatically entered into the entry box. To enter a label you must enter the alpha string into the box manually. If an invalid label was entered a Valid Label Selection Window will be displayed to allow you to change your label selection.

Enable Trace After Line

Purpose:
The Enable Trace After Line entry specifies the DeltaMax when to enable tracing. Only AFTER this line has been executed, will the DeltaMax start to look for the Trace Line.

Entry method:
The Trace Line entry box accepts either a source line number or an Executable Source Line Label. Checks are made on the validity of those entries. To enter a line number you can enter the number manually in the box or you can click on a line in the Source window. When you click on a line in the Source window the associated line number will be automatically entered into the entry box. To enter a label you must enter the alpha string into the box manually. If an invalid label was entered a Valid Label Selection Window will be displayed to allow you to change your label selection.

Valid Label Selection Window

The following figure shows the Valid Label Selection window that would be displayed if an invalid label entry was entered into the Trace Line and Enable Trace After Line entry boxes. Valid labels are Executable Source Line Labels.

Entry method:
You can type in a label that is shown on the list and then click on Accept. You can also double click on the label of choice.

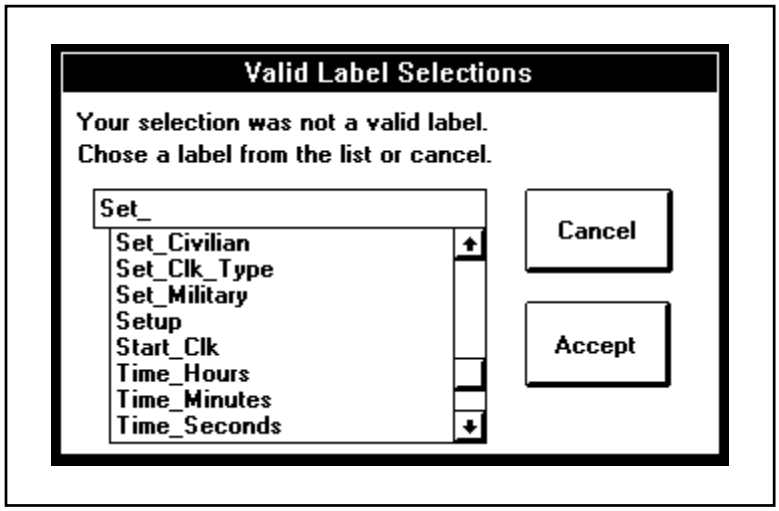


Figure 4.24 - Valid Label Selection Window

Executable Source Line Labels

This is a label that is tied to a line in the program that will be executed when the program runs. Labels for equates or data are not considered executable labels.

Read Trace

This menu item is not available on the Trace window.

In some cases a trace might not complete as expected. In the event this happens, but you still want to see what is currently in the DeltaMax's trace memory, select Read Trace. This feature will stop the active trace then access the current state of the trace memory in the DeltaMax. For this feature the trace "Settings" are ignored.

View Last Trace

This menu item is not available on the Trace window.

This feature will open the Trace window if not already open and display the last trace results.

Window Menu Item

To access the Window feature, use the [Alt + w] key combination or select Window from the Menu Item Bar. The windows that are currently open and which one is active will be displayed here. A selection to cascade the windows or to select which window is currently active can be made.

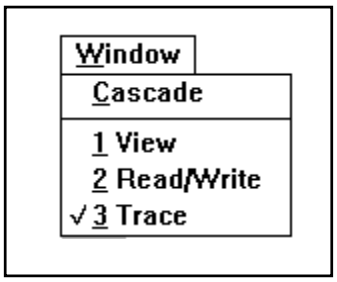


Figure 4.25 - Window Menu

Help Menu Item

To access the Help feature, use the [Alt + h] key combination or select Help from the Menu Item Bar. From this menu a selection to the desired help section can be made.

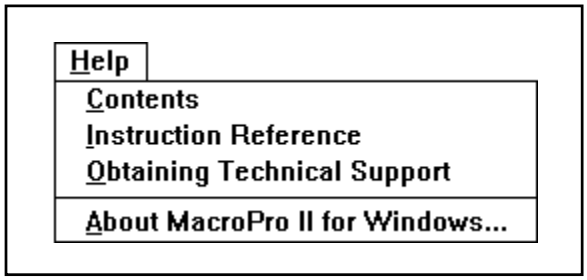


Figure 4.26 - Help Menu

Help

Help gives you access to the Help facility for the Analyzer. The F1 function key will execute this operation. Help for the window that is currently active in the analyzer will be displayed.

MACROPRO II™

FOR WINDOWS™

Section 6 - Instruction Reference
& Examples

INDUSTRIAL INDEXING SYSTEMS, INC.

SECTION 6 - INSTRUCTION REFERENCE

Table of Contents (Instructions)

Table of Contents (Examples)

Type Allowed

Abreviation	Definition
A	Address Label
FC	Float Constant
FE	Float Equate
FV	Float Variable
IC	Integer Constant
IE	Integer Equate
IV	Integer Variable
T	Text

analog_in

SYNTAX:

label analog_in controller#,channel#,value

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	Analog controller number	3	IE, IC
channel#	Input channel number	1 to 2.	IE, IC
value	A value containing the analog reading from the specified input channel number	-1023 to +2046	IV

DESCRIPTION:

Perform a read from the specified channel of the analog controller. The data is placed in 'value' after it is modified based on the channel's current slew rate limit and offset parameters. The modified data will have a range of -1024 to +2046 which is equivalent to 0 VDC to +5 VDC when the offset is 0.

The value read may be delayed up to 11 milliseconds due to access time through the update loop.

RETURNS:

Returns the current reading of the specified analog input channel plus any currently set offset value and limited by current slew rate setting.

analog_out

SYNTAX:

label analog_out controller#,channel#,value

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	controller id#	3	IE, IC
channel#	Channel to be used.	3	IE, IC
value	Value to output.	0 to 1023	IE, IC, IV

DESCRIPTION:

Modify the output value by the specified channel's current slew rate limit and offset parameters and then write to the specified channel. The specified channel update may be delayed for as long as 11 milliseconds due to the access time of the control loop.

Executing the **f_decel** macro instruction will slew the output channel to 0 volts.

The modified output data will have a range from 0 to +1023 which is equivalent to 0 VDC to +5 VDC. If an added offset would modify an **analog_out** outside, the output range is ignored.

Outputs are initialized to 0.0 volts on power-up. The analog output channel will slew to 0.0 volts when the program is stopped by the Analyzer.

RETURNS:

None.

analog_rt

SYNTAX:

label analog_rt controller#,channel#,value

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	controller id#	3	IE, IC
channel#	Channel to be used.	1 to 3	IE, IC
value	The slew rate limit.	0 to 1023	IE, IC, IV

DESCRIPTION:

Sets the specified channel slew rate limit in bits per 10 milliseconds. The slew rate value limits the rate at which a particular channel (input or output) can change. A slew rate of 1 is equivalent to a rate of change of 4.88 mV per 10 milliseconds.

RETURNS:

None.

analog_zo

SYNTAX:

label analog_zo controller#,channel#,value

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	controller id#	3	IE, IC
channel#	Channel to be used.	1 to 3	IE, IC
value	Offset value.	-1023 to +1023	IE, IC, IV

DESCRIPTION:

Sets the specified channel offset value in 4.88 millivolts per bit. Offset value is cleared on power-up. The analog output channel will slew to 0.0 volts when the program is stopped by the Analyzer. Zero offset values are cleared on power-up.

RETURNS:

None.

begin_cam

SYNTAX:

label begin_cam

PARAMETERS:

None.

DESCRIPTION:

Signals the start of a cam data area which will contain 8 bit (1 byte) values which represent incremental cam data. Each **begin_cam** instruction must have a corresponding **end_cam** instruction.

This instruction requires a 'label'.

EXAMPLE

```
cam_array   begin_cam
            cam         1,2,3,4,5
            cam         6,6,6,6,6
            cam         5,4,3,2,1
            end_cam
```

RETURNS:

None.

begin_cfg

SYNTAX:

label begin_cfg

PARAMETERS:

None

DESCRIPTION:

This is a directive to the MacroPro II™ Compiler.

The directive informs the compiler and eventually the Deltamax Operating System, that the following commands up to the end_cfg directive, are to be executed immediately following a program download to the DeltaMax.

RETURNS:

None.

begin_data

SYNTAX:

label begin_data

PARAMETERS:

None.

DESCRIPTION:

Signals the beginning of a data area which will contain 32-bit (4 byte) values.

This instruction requires a 'label'.

EXAMPLE

```
profile     begin_data
            data        100,200,15*4096
            data        0,200,40*4096
            end_data
```

RETURNS:

None.

blk_io_in

SYNTAX:

label blk_io_in input_flag#,value

PARAMETERS:

Name	Description	Range	Type Allowed
input_flag#	Starting flag number to read. Must be a multiple of 8.	0 or 8	IE, IC
value	Value of I/O flags to be read.		IV

DESCRIPTION:

The first eight bits in 'value' are used to store the state of the I/O modules, starting with 'input_flag#'.

EXAMPLE

If 'value' = 3, the first two inputs are ON.

If 'value' = 255, all eight input modules are ON.

RETURNS:

A four byte value with the LS byte representing the ON and OFF states of the eight I/O read. (Bit ON, I/O is ON)

blk_io_out

SYNTAX:

label blk_io_out output_flag#,value

PARAMETERS:

Name	Description	Range	Type Allowed
output_flag#	Starting flag number to modify.	16	IE, IC
value	Value of I/O flags to set.		IE, IC, IV

DESCRIPTION:

The first eight bits in 'value' are used to set or clear the eight outputs.

EXAMPLE

If 'value' = 3, the first two output modules will be turned ON.

If 'value' = 255, all eight output modules will be turned ON.

RETURNS:

None.

calc_cam_sum

SYNTAX:

label calc_cam_sum controller#,starting element,ending element

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	controller id #	1	IE, IC
starting element	Element # relative to axis memory zero.		IE, IC, IV
ending element	Element # relative to axis memory zero.		IE, IC, IV

DESCRIPTION:

Sums the values of cam elements in axis controller memory starting with 'starting element' and including 'ending element'. The starting and ending element numbers are relative to axis controller memory location zero.

The axis status flag **CALCULATION IN PROGRESS** will be ON until calculations are complete.

The calculated sum can be retrieved using the **get_cam_sum** instruction.

RETURNS:

None.

calc_unit_cam

SYNTAX:

label calc_unit_cam controller#,distance, # of elements, starting element

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	controller id#	1	IE, IC
distance	Total distance of the cam in bits.		IE, IC, IV
# of elements	Number of elements which make up the cam.		IE, IC, IV
starting element	The first element of the cam. This element is relative to axis memory location zero.		IE, IC,IV

DESCRIPTION:

Allows the user to define the shape of a cam and to have the axis controller calculate the cam data array. A table of co-efficients must first be transmitted, using the 'let' instruction, into the axis controller memory block, beginning at byte address 27,000. This table must be 130 elements in length.

The 'calc_unit_cam' instruction will then provide the total cam distance, the number of elements in the cam and the starting element for the cam (relative to axis memory location zero).

RETURNS:

None.

cam

SYNTAX:

label cam value,value,etc.

PARAMETERS:

Name	Description	Range
value	Incremental cam value.	-127 to +127

DESCRIPTION:

Specifies one-byte incremental values for an electronic cam.

Expressions are allowed. More than one cam value may be contained in a **cam** statement. By placing the **end_cam** statement at the end of the cam table, a cam table terminator (128) is automatically generated.

EXAMPLE

```
cam_array   begin_cam
            cam         1,2,3,4,5,6,7,8,9,10
            cam         10,10,10,10,10,10,10
            cam         10,9,8,7,6,5,4,3,2,1
            end_cam
```

RETURNS:

None.

cam_data

SYNTAX:

label cam_data controller#,data_label,master_scale,data_scale

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	controller id#	1	IE, IC
data_label	Program label of the cam data.		T
master_scale	Number of times to right shift the master position as it is received by the controller.	0 to 12	IE, IC

Master Scale Factor	Cam array advances every
12	1 full turn
11	1/2 turn
10	1/4 turn
9	1/8 turn
8	1/16 turn
7	1/32 turn
6	1/64 turn
5	1/128 turn
4	1/256 turn
3	1/512 turn
2	1/1024 turn
1	1/2048 turn
0	1/4096 turn

data_scale	Number of times to left shift each data element.	0 to 7	IE, IC

DESCRIPTION:

The **cam_data** instruction provides the controller with its' master scale, data scale and cam data array location.

When 'data_label' is dimensioned to reside within program memory, this instruction will also transfer the data array associated with 'data_label' to the controller.

When 'data_label' is dimensioned to reside directly on the volatile memory of the axis controller, no transfer of data will occur. Data can be transferred to the controller memory using the **let** or **let_byte** instructions before and/or after the **cam_data** instruction has been issued. This instruction then serves as a pointer to that data array. More than 1 data array can be dimensioned for the controller.

RETURNS:

None.

can_set

SYNTAX

label can_set arg1,arg2,arg3

PARAMETERS:

Name	Description	Range	Type Allowed
arg1	DeviceNet ID	0 to 63, -1	IV, IC
arg2	DeviceNet Baudrate in kbaud	125,250,500, -1	IV, IC
arg3	Protocol setting	2	IV, IC

DESCRIPTION:

Configures the DeviceNet MacID and Baudrate.

If arg1 and/or arg2 are set to -1 then that DeviceNet parameter may be set over the DeviceNet network. The DeltaMax may then be configured by a software tool such as DeviceNet Manager by Allen-Bradley. Once the parameter is configured it is stored in Nonvalitile memory so that it is remembered by the controller from one powerup to the next.

The DeviceNet ID and Baudrate may be set back to the default values of 63 and 125Kbaud respectively, by RESETing the DeltaMax thru the MACROPRO_II Analyzer then cycling power to the unit.

A can_set executed after a CanBus fault flag (flag# 129) will attempt to clear the fault and bus-off condition so that communications may continue.

RETURNS:

None.

case

SYNTAX

label case num

PARAMETERS:

Name	Description	Range	Type Allowed
num	An integer value.	-32768 to 32767	IC

DESCRIPTION:

Used with the **select** instruction to designate a branch address. Each **case** value must be unique.

EXAMPLE

```
select      num
        case    1
        .
        .
        exit_select

        case    2
        .
        .
        exit_select

        default
        .
        .
        exit_select

end_select
```

RETURNS:

None.

clr_all_swi

SYNTAX:

label clr_all_swi

PARAMETERS:

None.

DESCRIPTION:

Disables and clears all 32 (0 through 31) software interrupts.

RETURNS:

None.

clr_bit

SYNTAX:

label clr_bit bit#,value

PARAMETERS:

Name	Description	Range	Type Allowed
bit#	Number of the bit within the 4-byte value to be cleared to off (logic 0).	0 to 31	IE, IC, IV
value	The 4-byte area in memory affected by this instruction.		IV

DESCRIPTION:

The specified bit will be cleared (logic 0). If that bit has been previously cleared, it will remain cleared. If that bit was previously set (logic 1), it will now be cleared.

This instruction has no effect on the remaining bits of this 4-byte value.

This instruction will be ignored if 'bit#' is outside the range of 0 to 31.

RETURNS:

None.

clr_dnet

SYNTAX:

label clr_dnet flag#

PARAMETERS:

Name	Description	Range	Type Allowed
flag#	DeviceNet flag number to be cleared.	56 to 71	IE, IC

DESCRIPTION:

Allows setting of a Devicenet flag from 56-71, sets the value of the flag boolean level false or zero.

RETURNS:

None.

clr_flag

SYNTAX:

label clr_flag user_flag#

PARAMETERS:

Name	Description	Range	Type Allowed
user_flag#	Number of the user flag to be cleared.	208 to 255	IE, IC

DESCRIPTION:

Clears the specified user flag.

RETURNS:

None.

clr_local

SYNTAX:

label clr_local controller#

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	controller id #	1 or 2	IE, IC

DESCRIPTION:

Clears the current local zero position and causes the specified axis controller to use the current global zero as the absolute zero position.

RETURNS:

None.

clr_pls

SYNTAX:

label clr_pls controller#,module#

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	controller id#	2	IE, IC
module#	PLS output module.	16 to 23	IE, IC

DESCRIPTION:

Clears the specified PLS module and allows that module to be used as a normal output. After this instruction is issued the PLS must be redefined to use as an PLS module again.

RETURNS:

None.

clr_swi

SYNTAX:

label clr_swi interrupt#

PARAMETERS:

Name	Description	Range	Type Allowed
interrupt#	Interrupt number.	0 to 31	IE, IC

DESCRIPTION:

Clears a previously defined software interrupt.

RETURNS:

None.

clr_sys_error

SYNTAX:

label clr_sys_error flag#

PARAMETERS:

Name	Description	Range	Type Allowed
flag#	dedicated system error flag	144 to 159	IE, IC

DESCRIPTION:

The DeltaMax controller will verify that data used with a program instruction is valid at run time. When it is determined that an attempt to use invalid data has been made, a dedicated flag will be enabled. That flag is known as a system error flag.

This flag is set in order for the program to monitor and detect a problem with data used by a certain instruction. The program could monitor the state of this flag through the use of the **if_flag_on** or **if_flag_off** instructions or through the use of the software interrupts.

The system error flag will remain enabled until one of three events occur:

1 - a controller RESET has been executed

2 - the controller power has been cycled from off to on

3 - the flag is cleared within the program

The program instruction **clr_sys_error** will clear the system error flag.

RETURNS:

None.

data

SYNTAX:

label data value,value,etc.

PARAMETERS:

Name	Description	Range
value	32-bit constant	-2,147,483,648 to +2,147,483,647

DESCRIPTION:

Specifies 32-bit values. More than 1 value may be contained in a 'data' statement.

EXAMPLE

```
profile     begin_data
            data        100,200,4096
            data        0,200,4096
            end_data
```

RETURNS:

None.

data_scale

SYNTAX:

label	data_scale	controller#,dscale

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	controller id #	1	IE, IC
dscale	cam related master scale	0 to 7	IE, IC, IV

DESCRIPTION:

The **cam_data** instruction provides cam setup information to the controller. This includes the location of the data which defines the cam, the master scale factor and the data scale factor.

Due to the amount of information it provides, the **cam_data** instruction is a relatively lengthy instruction to execute, compared to most instructions. The **data_scale** instruction can be used when the only information to be changed regarding cam processing is the data scale factor.

The data scale factor is used to scale the distance per cam element.

A data scale factor of '0' indicates that the contents of each cam element would be scaled by a factor of 2^0 (which is 1). A data scale factor of '7' indicates that the contents of each cam element would be scaled by a factor of 2^7 (which is 128).

For example, the total distance defined by a cam is 8000 bits. Using a data scale of '1', the total distance moved by this same cam would be 16,000 bits, since each cam element is scaled by 2^1 (which is 2).

The **data_scale** instruction can be used to initiate a new data scale factor (cam element execution rate) without the need to pass additional information to the controller in the form of the **cam_data** instruction.

RETURNS:

None.

declare

SYNTAX:

label declare mode

PARAMETERS:

Name	Description
mode	This entry can be only ON or OFF.

DESCRIPTION:

This is a directive to the MacroPro II™ Compiler.

If 'mode' is ON, the Compiler will expect that all values, text strings, equates etc. will be declared by the programmer. They will not be automatically created by the Compiler.

If 'mode' is OFF, the Compiler will automatically create data space for those values used but not declared.

The Compiler will assume that 'mode' is OFF if this instruction is not included in the program.

RETURNS:

None.

default

SYNTAX:

label default

PARAMETERS:

None.

DESCRIPTION:

Used with the **select** instruction to designate a group of statements which are executed if none of the **case** statements produce a match.

EXAMPLE

```
select   x
         case    1
         .
         .
         exit_select

         case    2
         .
         .
         exit_select

         default
         exit_select

end_select
```

RETURNS:

None.

delta_comp

SYNTAX:

label delta_comp controller#,AJ2,AJ3,AJ4,AJ7,AJ8,AJ9

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	controller id#	1	IE,IC
AJ2	load iniertia magnification	0.0 to 100	IE,IC,IV,FE,FC,FV
AJ3	high speed response	0.1 to 20.0	IE,IC,IV,FE,FC,FV
AJ4	position gain	1 to 200	IE,IC,IV,FE,FC,FV
AJ7	gain reduction while stopped	0 to 10000	IE,IC,IV,FE,FC,FV
AJ8	feed forward gain	0.0 to 2.0	IE,IC,IV,FE,FC,FV
AJ9	current command filter	100 to 20000	IE,IC,IV,FE,FC,FV

DESCRIPTION:

Sets digital compensation values at the Delta Drive. When executed, the values are transmitted to the Delta Drive and are effective immediately. When AJ9 (current command filter) is changed, power must be cycled to the Delta Drive for this parameter to take effect.

The defaults for the AJ Parameters are as follows:

Name	Default
AJ2	1.0
AJ3	1.0
AJ4	30
AJ7	0
AJ8	1.0
AJ9	6000

The subsequent execution of a ***delta_comp*** instruction in the program will cause the AJ Parameters to be set as needed for your application.

RETURNS:

None.

digi_comp

SYNTAX:

label digi_comp controller #,gain,value,value

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	controller id#	1	IE, IC
gain	Used to adjust the size of the allowed following error window.	2 to 32768	IE, IC, IV
value	not used	0	IE, IC
value	not used	0	IE, IC

DESCRIPTION:

The default following error window for DeltaMax systems is +/-2048 bits. This corresponds to the default 'gain' of 16.

Changing the 'gain' value will effect the size of the following error window. For example:

- a 'gain' of 32 will result in a following error window of +/-1024 bits.

- a 'gain' of 8 will result in a following error window of +/-4096 bits.

The parameters 'value' are unused.

RETURNS:

None.

dim

SYNTAX:

label	dim	size
	or	
label	dim	controller #,size

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	controller id#	1	IE, IC
size	Number of 32-bit storage locations to reserve.		IE, IC

DESCRIPTION:

This statement allocates the indicated number of storage locations to the name 'label'. This instruction may be used to allocate all or part of the 28,000 bytes of the controller memory.

When controller #1 is used in the 'dim' instruction, the array will be allocated within the 28,000 bytes of volatile memory of controller #1. The size may range from 1 to 7,000.

When the controller# is not used in the 'dim' instruction, the array will be allocated within the non-volatile program/data memory. The size may range from 1 to 4,000.

This instruction requires a 'label'.

RETURNS:

None.

disable_hwi

SYNTAX:

label disable_hwi controller#

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	controller id#	1	IE, IC

DESCRIPTION:

Terminates scanning of the hardware interrupt signal in the axis controller. Also clears the appropriate **HARDWARE INTERRUPT ARMED** status in the controller.

RETURNS:

None.

disable_swi

SYNTAX:

label disable_swi

PARAMETERS:

None.

DESCRIPTION:

Disables software interrupt processing. Previously defined system interrupts will not be cleared, merely disabled.

RETURNS:

None.

dnet_flag_pm

SYNTAX:

label dnet_flag_pm dnet_flag,text_label

PARAMETERS:

Name	Description	Range	Type Allowed
dnet_flag	DeviceNet I/O flag number	32 to 71	IC
text_label	Label of a text string	16 characters maximum	T

DESCRIPTION:

This instruction must reside in a program between a begin_cfg and end_cfg instruction at the top of a macroprogram.

Assigns a parameter to the DeviceNet programmable Parameter Object of the controller. The parameter assigned is the instance of the Flag Object associated with the dnet_flag argument of the instruction. During instruction execution the MacroProgram variable shall be given the default value of the instruction.

Instances of the Parameter Object are arranged in the order of which they are assigned in a program. No more than 16 assignments may be made in a program, in that their can only be 16 instances of the Parameter Object. That includes assignments from instructions **dnet_intg_pm** and **dnet_flt_pm**.

RETURNS:

None.

dnet_flt_pm

SYNTAX:

label dnet_flt_pm value,text_label,min,max,default

PARAMETERS:

Name	Description	Range	Type Allowed
value	The declared integer value to use for a DeviceNet parameter.		IF
text_label	Label of a text string to describe the parameter.	16 characters maximum	T
min	Minimum intended value for the assigned parameter.	IEE754 double floating-point	FC
max	Maximum intended value for the assigned parameter.	IEE754 double floating-point	FC
default	Default value for the assigned parameter.	IEE754 double floating-point	FC

DESCRIPTION:

This instruction must reside in a program between a begin_cfg and end_cfg instruction at the top of a macroprogram.

Assigns a parameter to the DeviceNet programmable Parameter Object of the controller. The parameter assigned is the instance of the Float Object associated with the float value argument of the instruction. During instruction execution the MacroProgram variable shall be given the default value of the instruction.

Instances of the Parameter Object are arranged in the order of which they are assigned in a program. No more than 16 assignments may be made in a program, in that their can only be 16 instances of the Parameter Object. That includes assignments from instructions **dnet_flag_pm** and **dnet_intg_pm**.

IMPORTANT: The **dnet_range** instruction should be executed prior to this instruction, so that the parameter may be written to over the DeviceNet port if desired. Otherwise the parameter shall have a READ ONLY attribute.

RETURNS:

None.

dnet_int_pm

SYNTAX:

label dnet_int_pm value,text_label,min,max,default

PARAMETERS:

Name	Description	Range	Type Allowed
value	The declared integer value to use for a DeviceNet parameter.		IV
text_label	Label of a text string to describe the parameter.	16 characters maximum	T
min	Minimum intended value for the assigned parameter.	-2147483647 to 2147483647	IC
max	Maximum intended value for the assigned parameter.	-2147483647 to 2147483647	IC
default	Default value for the assigned parameter.	-2147483647 to 2147483647	IC

DESCRIPTION:

This instruction must reside in a program between a begin_cfg and end_cfg instruction at the top of a macroprogram.

Assigns a parameter to the DeviceNet programmable Parameter Object of the controller, see IB-19B005. The parameter assigned is the instance of the Integer Object associated with the integer value argument of the instruction. During instruction execution the MacroProgram variable shall be given the default value of the instruction.

Instances of the Parameter Object are arranged in the order of which they are assigned in a program. No more than 16 assignments may be made in a program, in that their can only be 16 instances of the Parameter Object. That includes assignments from instructions **dnet_flag_pm** and **dnet_flt_pm**.

IMPORTANT: The **dnet_range** instruction should be executed prior to this instruction, so that the parameter may be written to over the DeviceNet port if desired. Otherwise the parameter shall have a READ ONLY attribute.

RETURNS:

None.

dnet_p_code

SYNTAX

label dnet_p_code product_code,text_label

PARAMETERS:

Name	Description	Range	Type Allowed
product_code	DeviceNet product code number in the Deltamax Identity Object.	1024 to 1279	IC
text_label	Label of a text string to describe the deltamax application program.		T

DESCRIPTION:

This instruction must reside in a program between a begin_cfg and end_cfg instruction at the top of a macroprogram.

Assigns a “DeviceNet Product Code” in the Identity Object of the controller. It also will change the “DeviceNet Product Type” in the Identity Object of the controller from 0Hex to 300Hex, so that the Product Type changes from “Generic” to “vendor_specific”.

The instruction is intended for giving different controllers on the same DeviceNet network unique identities. This is especially helpful when the controllers have different programs and application parameters to accomplish different task in a system having one DeviceNet Network.

RETURNS:

None.

dnet_range

SYNTAX

label dnet_range argument_1, argument_2

PARAMETERS:

Name	Description	Range	Type Allowed
argument_1	lower boundary	any variable label within the macroprogram	
argument_2	upper boundary	variable label in a sequencially higher order in memory than argument_1	

DESCRIPTION:

Allows the maroprogramer to set limits on what variables in the macroprogram may be written to by the DeviceNet port.

IMPORTANT: The **dnet_range** instruction should be executed prior to the parameter object programming instructions **dnet_flt_pm** and **dnet_int_pm**, so that the parameter may be written to over the DeviceNet port if desired.

RETURNS:

None.

drive_off

SYNTAX:

label drive_off controller#

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	controller ID #	1	IE, IC

DESCRIPTION:

Resets all motor fault conditions and puts the axis controller into a passive position sensing mode. Turns the servo amplifier off and disables the following error check.

RETURNS:

None.

drive_on

SYNTAX:

label drive_on controller#

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	controller id#	1	IE, IC

DESCRIPTION:

Reset all motor fault conditions and turns on the servo amplifier. The controller must be enabled at least once prior to executing any motion instructions. The controller initializes to 'reset mode' on power-up.

RETURNS:

None.

enable_hwi

SYNTAX:

label enable_hwi

PARAMETERS:

The instruction immediately following **enable_hwi** will be used.

DESCRIPTION:

This instruction, when used with supported 'hardware interrupt' instructions, will enable scanning at the specified motion controller for the 'hardware interrupt' signal to be activated.

When the 'hardware interrupt' signal is detected, the specified instruction will immediately be executed. The MacroPro II™ Compiler will associate the instruction that follows 'enable_hwi' as the instruction to be executed.

The following instructions are currently supported:

enable_hwi over_draw	controller id#,speed,limit,distance
enable_hwi trap_pos	controller id#
enable_hwi ratio	controller id#,ratio
enable_hwi lock	controller id#,lock type
enable_hwi position	controller id#,abs_position
enable_hwi index	controller id#,distance
enable_hwi exec_profile	controller id#
enable_hwi f_decel	controller id#

The **HARDWARE INTERRUPT ARMED** status in the controller will be on after this instruction has been executed.

RETURNS:

None.

enable_swi

SYNTAX:

label enable_swi

PARAMETERS:

None.

DESCRIPTION:

Enables software interrupt processing.

RETURNS:

None.

end_cam

SYNTAX:

label end_cam

PARAMETERS:

None.

DESCRIPTION:

Signals the end of a data area containing incremental cam data. Each **begin_cam** instruction must have a corresponding **end_cam** instruction.

Automatically places an end of cam value (80 hex) at the end of the cam array.

EXAMPLE

```
cam_array   begin_cam
            cam         1,2,3,4,5
            cam         6,6,6,6,6
            cam         5,4,3,2,1
            end_cam
```

RETURNS:

None.

end_cfg

SYNTAX:

label end_cfg

PARAMETERS:

None

DESCRIPTION:

This is a directive to the MacroPro II™ Compiler.

The directive informs the compiler and eventually the Deltamax Operating System, to stop executing configuration commands.

RETURNS:

None.

end_data

SYNTAX:

label end_data

PARAMETERS:

None.

DESCRIPTION:

Signals the end of a data array area. Each **begin_data** instruction must have a corresponding **end_data** instruction.

EXAMPLE

```
profile     begin_data
            data        100,200,4096
            data        0,200,4096
            end_data
```

RETURNS:

None.

end_select

SYNTAX:

label end_select

PARAMETERS:

None.

DESCRIPTION:

Used to end a group of **select/case** statements. Each **select** statement must have a corresponding **end_select** statement.

RETURNS:

None.

equ

SYNTAX:

label equ constant

PARAMETERS:

Name	Description	Type Allowed
constant	An integer or floating point number or expression.	IE, IC, FE, FC

DESCRIPTION:

The **equ** instruction assigns a symbol to a number or mathematic expression.

The MacroPro II™ Compiler replaces each occurrence of 'label' with the assigned number.

This instruction requires a 'label'.

RETURNS:

None.

exec_profile

SYNTAX:

label exec_profile controller#

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	Controller id#	1 or 2	IE, IC

DESCRIPTION:

Executes a previously defined piece-wise profile. While the profile is executing, the **get_pstat** instruction may be used to determine the profile segment currently running.

This instruction sets the controller status flags **BUSY** and **INDEXING**. These flags remain set until a **f_decel** instruction causes the motor to reach zero speed or the profile is completed.

This instruction will be ignored if no valid data has been transmitted to the controller via the **prep_profile** instruction.

This instruction will be ignored if the controller has not received a **drive_on** instruction, or if the controller is busy executing a motion instruction. If the instruction cannot be executed, the axis status flag **COMMAND INVALID IN THIS STATE** will be set. This flag will automatically be cleared by the controller at the next communication between the main processor and the axis controller.

RETURNS:

None.

exit_select

SYNTAX:

label exit_select

PARAMETERS:

None.

DESCRIPTION:

Used to end a group of statements being executed within a particular **case** statement. Causes program control to transfer to the statement following the **end_select** statement. Each **case** statement must have a corresponding **exit_select** statement.

EXAMPLE

```
select
        case        1
        .
        exit_select

        case        2
        .
        exit_select

        default
        .
        exit_select

end_select
```

RETURNS:

None.

float

SYNTAX:

label float

PARAMETERS:

None.

DESCRIPTION:

This statement assigns a 64 bit storage location, or value, to the name 'label'. The storage location will be initialized to zero when the program is loaded into the Controller.

This instruction requires a 'label'.

RETURNS:

None.

float_dim

SYNTAX:

label float_dim size

PARAMETERS:

Name	Description	Range	Type Allowed
size	Number of 64-bit storage locations to reserve.	1 to 2000	IE, IC

DESCRIPTION:

This statement allocates the indicated number of storage locations to the name 'label'.

This instruction requires a 'label'.

RETURNS:

None.

f_decel

SYNTAX:

label f_decel controller#

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	controller id#	1 or 2	IE, IC

DESCRIPTION:

The specified controller will be commanded to stop its motion at the last set accel/decel rate. If its motion is currently stopped, **f_decel** has no effect.

The controller status flag **FORCED DECEL IN PROGRESS** is set by this instruction. It remains set until the controller's motion reaches zero speed.

A mode zero unlock will occur if the controller is currently in a master/slave lock.

A **f_decel** instruction is automatically sent to all controllers whenever a **sys_fault** or a **sys_return** instruction is executed.

RETURNS:

None.

get_act_spd

SYNTAX:

label get_act_spd controller#,value

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	controller id#	1 or 2	IE, IC
value	Current motion controller speed in RPM.(4096 bits per revolution are assumed)		IV

DESCRIPTION:

Returns the actual speed at the motor shaft of the specified motion controller (in RPM) into 'value'.

RETURNS:

Actual axis speed.

get_angle

SYNTAX:

label get_angle controller#,value

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	controller ID #	1	IE, IC
value	The current angle of the master axis driving the controller in master/slave lock.		IV

DESCRIPTION:

Returns the current angle of the master axis driving the axis controller in master/slave lock.

RETURNS:

Master angle

get_cam_cnt

SYNTAX:

label get_cam_cnt controller#,value

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	controller id#	1	IE, IC
value	The returned cam counter.		IV

DESCRIPTION:

Returns the number of executions of the currently executing cam. The cam counter is incremented when the cam pointer rolls over from last element to first element, and is decremented each time the cam pointer rolls over from first element to last element.

RETURNS:

The number of times this cam has been executed.

get_cam_end

SYNTAX:

label get_cam_end controller#,value

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	controller id#	1	IE, IC
value	The last element in the current cam.		IV

DESCRIPTION:

Returns the ending location of the cam currently being executed, relative to the axis memory location zero.

RETURNS:

Returns the element number of the last value in the current cam.

get_cam_ptr

SYNTAX:

label get_cam_ptr controller#,value

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	controller id#	1	IE, IC
value	The cam element currently being referenced.		IV

DESCRIPTION:

Stores the current cam array element relative to the beginning of the cam into 'value'..

RETURNS:

The number of the current cam array element.

get_cam_strt

SYNTAX:

label get_cam_strt controller#,value

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	controller id#	1	IE, IC
value	The first element in the current cam.		IV

DESCRIPTION:

Retrieves the starting location of the cam currently executed, relative to axis controller memory location zero.

RETURNS:

The first element of the current cam relative to axis controller memory zero.

get_cam_sum

SYNTAX:

label get_cam_sum controller#,value

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	controller id#	1	IE, IC
value	Returned sum of cam elements.		IV

DESCRIPTION:

Returns the result of the last **calc_cam_sum** instruction in 'value'. This instruction may be executed after the instruction **calc_cam_sum** has been executed, and the axis status flag **CALCULATION IN PROGRESS** has gone from ON to OFF.

RETURNS:

The signal value equal to the sum of the elements as specified by the '**calc_cam_sum'** instruction.

get_com

SYNTAX:

label get_com controller#,value

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	controller id#	1 or 2	IE, IC
value	Returned position.		IV

DESCRIPTION:

Gets the absolute commanded motor position of the specified axis controller and places this position in 'value'. Position is a signed 32-bit number.

RETURNS:

Axis controllers commanded position.

get_fol_err

SYNTAX:

label get_fol_err controller#,value

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	controller id#	1	IE, IC
value	The current following error angle.		IV

DESCRIPTION:

Returns the current following error angle. This value is the difference between the current commanded position and the current actual position and is represented as a signed value with a normal range of ± 4095. Altering the digital gain to less than 16 may result in a value which exceeds normal range. CCW will return negative values, and CW motion will return positive values.

RETURNS:

Returns the difference between the current axis controller commanded and actual positions.

get_for_ang

SYNTAX:

label get_for_ang controller#,channel#,value

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	controller id#	2	IE, IC
channel#	The fiber optic channel#.	1	IE, IC
value	The current fiber optic angle.		IV

DESCRIPTION:

Returns the current fiber optic angle for the specified channel.

RETURNS:

Fiber optic angle. This value is a signed 32-bit number with the range -2,147,483,648 to +2,147,483,647.

get_for_spd

SYNTAX:

label get_for_spd controller#,channel#,value

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	controller id#	2	IE, IC
channel#	The fiber optic channel#.	1	IE, IC
value	The current speed in RPM.		IV

DESCRIPTION:

Returns the current speed in RPM from the device connected to the fiber optic channel. This speed is based on 4096 bits per motor revolution at this device.

RETURNS:

Speed of the device connected to the fiber optic channel. This value is a signed 16-bit number with the range -32768 to +32767.

get_map

SYNTAX:

label get_map value

PARAMETERS:

Name	Description	Type Allowed
value	The currently defined master angle bus configuration.	IV

DESCRIPTION:

Queries the current definition of the master angle bus communication configuration defined by the last **set_map** instruction.

RETURNS:

Returns the current master angle bus configuration. 'Value' may be interpreted as seen in Figure 3.21.

get_map_stat

SYNTAX:

label get_map_stat value

PARAMETERS:

Name	Description	Type Allowed
value	The current master angle bus status.	IV

DESCRIPTION:

MAP is an acronym for Master Angle Passing.

RETURNS:

Status of last **set_map** instruction executed.

Status	Description
0	Valid 'map' value.
1	More than 1 talker on Bus 'A'.
2	More than 1 talker on Bus 'B'.
3	Axis controller is receiving from Bus 'A' and 'B'.
4	Illegal talkers and/or listeners on Bus 'A'.
5	Illegal talkers and/or listeners on Bus 'B'.

get_mcf

SYNTAX:

label get_mcf controller#,value

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	controller id#	2	IE, IC
value	The currently defined Multi Function Controller configuration.		IV

DESCRIPTION:

MCF is an acronym for Multi Function Controller.

RETURNS:

Returns the current configuration of the Multi Function Controller, as defined by the last valid **set_mcf** instruction.

This 4-byte value is used as shown in Figure 3.22.

get_mcf_stat

SYNTAX:

label get_mcf_stat value

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	controller id#	2	IE, IC
value	The current multi function controller status.		IV

DESCRIPTION:

MCF is an acronym for Multi Function Controller.

RETURNS:

Status of last **set_mcf** instruction executed.

Status	Description
0	Valid 'mcf' value.
1	Too many 'sources' for PLS.
2	Too many 'sources' on Bus 'A'.
3	Too many 'sources' on Bus 'B'.

get_ms_time

SYNTAX:

label get_ms_time value

PARAMETERS:

Name	Description	Range	Type Allowed
value	Returned time	0 to 2,147,483,648	IV

.

DESCRIPTION:

Return the current system time in 'value'. The 'value' is an unsigned 32-bit data value. The resolution of the timer is 1 millisecond.

For example, a ‘value’ of 10000 would indicate that 10000 milliseconds (10 seconds) has elapsed from the moment the controller was powered up.

The system time will be reset to 0 during system power up.

RETURNS:

Current system time from power up.

get_pls_ang

SYNTAX:

label get_pls_ang controller#,pls_angle

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	controller id #	2	IE, IC
pls_angle	Angle used to drive the PLS modules.	0 to 8,388,607	IV

DESCRIPTION:

The **get_pls_ang** instruction returns the current angle used by the PLS function to determine the state of all defined PLS modules (ON or OFF).

This value will always be a number within the range specified by the **set_pls_cnt** instruction. If a **set_pls_cnt** instruction was not previously executed, the default range is 4096.

For example, the following **set_pls_cnt** instruction defines a PLS range of 40000 bits:

label **set_pls_cnt** **2,40000**

Any subsequent **get_pls_ang** instruction will return a PLS angle between 0 and 39999.

RETURNS:

A PLS angle within the range established by the **set_pls_cnt** instruction.

get_pls_mask

SYNTAX:

label get_pls_mask controller#,value

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	controller id#	2	IE, IC
value	The currently defined PLS mask value.		IV

DESCRIPTION:

PLS is an acronym for Programmable Limit Switch.

Returns the currently defined PLS mask value into 'value', as defined by the last **set_pls_mask** instruction.

Only the high order 2 bytes are used.

EXAMPLE:

If bit 16 of 'value' is set, PLS #16 is enabled and if bit 31 of 'value' is set, PLS #31 is enabled. Bits 0 to 15 are unused.

RETURNS:

The currently defined pls mask.

get_pls_out

SYNTAX:

label get_pls_out controller#,value

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	controller id#	2	IE, IC
value	The current state of the PLS output modules.		IV

DESCRIPTION:

PLS is an acronym for Programmable Limit Switch.

Returns the current state of the PLS output modules into 'value'. A bit on (logic 1) indicates that the associated module is ON. A bit off (logic 0) indicates that the associated output is OFF.

Only the high order 2 bytes are used.

Bits 16 to 23 are the hardware PLS flags.

Bits 24 to 31 are the internal PLS flags.

RETURNS:

Current state of PLS outputs.

get_pos

SYNTAX:

label get_pos controller#,value

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	controller id#	1 or 2	IE, IC
value	returned position.		IV

DESCRIPTION:

Gets the absolute actual motor position of the specified controller and places this position in 'value'. The position is a signed 32-bit value.

RETURNS:

The current actual motor position.

get_pstat

SYNTAX:

label get_pstat controller#,status

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	controller id#	1 or 2	IE, IC
status	Returned status.		IV

DESCRIPTION:

Returns the calculation or running status of a Piecewise Profile.

A. After a profile calculation has been performed, **get_pstat** is used to check the results of the calculations. If there were no errors, 'status' is zero. If any part of the calculations failed, then **get_pstat** will return a calculation error code and the number of the profile segment that caused the error as follows:

Least significant byte: profile segment number in error
Next significant byte: error code (see below)

B. While a profile is executing, **get_pstat** returns the number of the profile segment that is being executed.

RETURNS:

Error Codes:

Code	Meaning
1	Attempt to change profile direction.
3	Insufficient distance for specified speed and/or acceleration.

The following example represents a 'status' where the profile segment number in error is 10 and the error code for that segment is 3:

status: 778 (decimal) status: 0000 030A (hexadecimal)

get_time

SYNTAX:

label get_time value

PARAMETERS:

Name	Description	Type Allowed
value	Returned time.	IV

DESCRIPTION:

Return the current system time in 'value'. The 'value' is an unsigned 32-bit data value. The resolution of the timer is 5 milliseconds.

The timer will reset to 0 during system power up.

RETURNS:

Current system time from power-up.

get_trap_pos

SYNTAX:

label get_trap_pos controller#,value

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	controller id#	1	IE, IC
value	Returned position.	(-524288*4096) to (+524288*4096)-1	IV

DESCRIPTION:

Returns the actual motor position of the controller (saved by executing the last **trap_pos** instruction) into 'value'.

RETURNS:

Returns last trapped position.

gosub

SYNTAX:

label gosub subroutine_label

PARAMETERS:

Name	Description	Type Allowed
subroutine_label	Program label of the subroutine.	A

DESCRIPTION:

Branches to the specified address. When a **return_sub** instruction is executed the program branches to the instruction following the **gosub** instruction.

The Delta Max provides 20 levels of subroutine nesting. The **STACK OVERFLOW** status flag will be set if more than 20 levels are used.

RETURNS:

None.

goto

SYNTAX:

label goto address_label

PARAMETERS:

Name	Description	Type Allowed
address_label	Label where execution will continue.	A

DESCRIPTION:

Branches to the specified address label.

RETURNS:

None.

if

SYNTAX:

label if compare1 operator compare2,address_label

PARAMETERS:

Name	Description	Type Allowed
compare1	Value to be compared.	IE, IC, IV, FE, FC, FV
operator	Type of comparison to be performed.	

Operator	Description
=	Equal to
<>	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

compare2	Value to be compared.	IE, IC, IV, FE, FC, FV
address_label	Branch address if comparison is true.	A

DESCRIPTION:

Performs arithmetic comparison and causes branching to the specified address if the comparison is true.

If the comparison is false, no branching occurs and execution continues with the next instruction.

RETURNS:

None.

if_bit_clr

SYNTAX:

label if_bit_clr bit#,value,address_label

PARAMETERS:

Name	Description	Range	Type Allowed
bit#	Bit number of the 4-byte value to be tested.	0 to 31	IE, IC, IV
value	The 4-byte value to be tested.		IE, IC, IV
address_label	Branch address.		A

DESCRIPTION:

Branch to the specified address if the specified bit is off (logic 0).

If the bit tested is on (logic 1), no branching occurs and execution continues with the next instruction.

This instruction will be ignored if 'bit#' is outside the range of 0 to 31.

RETURNS:

None.

if_bit_set

SYNTAX:

label if_bit_set bit#,value,address_label

PARAMETERS:

Name	Description	Range	Type Allowed
bit#	Bit number of the 4-byte value to be tested.	0 to 31	IE, IC, IV
value	The 4-byte value to be tested.		IE, IC, IV
address_label	Branch address.		A

DESCRIPTION:

Branch to the specified address if the specified bit is on (logic 1).

If the bit tested is off (logic 0), no branching occurs and execution continues with the next instruction.

This instruction will be ignored if 'bit#' is outside the range of 0 to 31.

RETURNS:

None.

if_char

SYNTAX:

label if_char port#,address_label

PARAMETERS:

Name	Description	Range	Type Allowed
port#	Communications port number.	2	IE, IC
address_label	Branch address.		A

DESCRIPTION:

Branches to the specified address if characters are sensed at the specified port.

If no characters are sensed, execution continues with the next instruction.

RETURNS:

None.

if_dnet_off

SYNTAX:

label if_dnet_off flag#,address_label

PARAMETERS:

Name	Description	Range	Type Allowed
flag#	DeviceNet flag number to be tested.	32 to 71	IE, IC
address_label	instruction label within macroprogram.		

DESCRIPTION:

Allows condition checks on the DeviceNet flags 32-71, if the flag is off the branch is executed.

RETURNS:

None.

if_dnet_on

SYNTAX:

label if_dnet_on flag#,address_label

PARAMETERS:

Name	Description	Range	Type Allowed
flag#	DeviceNet flag number to be tested.	32 to 71	IE, IC
address_label	instruction label within macroprogram.		

DESCRIPTION:

Allows condition checks on the DeviceNet flags 32-71, if the flag is on the branch is executed.

RETURNS:

None.

if_flag_off

SYNTAX:

label if_flag_off user_flag#,address_label

PARAMETERS:

Name	Description	Range	Type Allowed
user_flag#	Flag number to test.	208 to 255	IE, IC
address_label	Branch address.		A

DESCRIPTION:

Branch to the specified address if the specified user flag is off.

If the user flag is on, no branching occurs and execution continues with the next instruction.

RETURNS:

None.

if_flag_on

SYNTAX:

label if_flag_on user_flag#,address_label

PARAMETERS:

Name	Description	Range	Type Allowed
user_flag#	Flag number to test.	208 to 255	IE, IC
address_label	Branch address.		A

DESCRIPTION:

Branch to the specified address if the specified user flag is on.

If the user flag is not on, no branching occurs and execution continues with the next instruction.

RETURNS:

None.

if_io_off

SYNTAX:

label if_io_off I/O flag#,address_label

PARAMETERS:

Name	Description	Range	Type Allowed
I/O flag#	Input or Output module number.	0 to 23	IE, IC
address_label	Branch address.		A

DESCRIPTION:

Branches to the specified address if the specified I/O flag is off.

If the I/O flag is on, branching does not occur and execution continues with the next instruction.

RETURNS:

None.

if_io_on

SYNTAX:

label if_io_on I/O flag#,address_label

PARAMETERS:

Name	Description	Range	Type Allowed
I/O flag#	Input or Output module number.	0 to 23	IE, IC
address_label	Branch address.		A

DESCRIPTION:

Branches to the specified address if the specified I/O flag is on.

If the I/O flag is off, branching does not occur and execution continues with the next instruction.

RETURNS:

None.

if_no_char

SYNTAX:

label if_no_char port#,address_label

PARAMETERS:

Name	Description	Range	Type Allowed
port#	Communications port number.	2	IE, IC
address_label	Branch address.		A

DESCRIPTION:

Branches to the specified address if no characters are sensed at the specified port.

If characters are sensed, execution continues with the next instruction.

RETURNS:

None.

if_stat_on

SYNTAX:

label if_stat_on status_flag #,address_label

PARAMETERS:

Name	Description	Range	Type Allowed
status_flag#	Controller status flag.	80 to 127	IE, IC
address_label	Branch address.		A

DESCRIPTION:

Branches to the specified address label if the controller status flag indicated is on.

If the controller status flag is off, branching does not occur and execution continues with the next instruction.

RETURNS:

None.

if_stat_off

SYNTAX:

label if_stat_off status_flag #,address_label

PARAMETERS:

Name	Description	Range	Type Allowed
status_flag#	Controller status flag.	80 to 127	IE, IC
address_label	Branch address.		A

DESCRIPTION:

Branches to the specified address label if the controller status flag indicated is off.

If the controller status flag is on, branching does not occur and execution continues with the next instruction.

RETURNS:

None.

if_tmr_off

SYNTAX:

label if_tmr_off timer_flag#,address_label

PARAMETERS:

Name	Description	Range	Type Allowed
timer_flag#	Timer flag number.	72 to 79	IE, IC
address_label	Branch address.		A

DESCRIPTION:

Branches to the specified address if the timer flag indicated is off.

If the timer flag is on, branching does not occur and execution continues with the next instruction.

RETURNS:

None.

if_tmr_on

SYNTAX:

label if_tmr_on timer_flag#,address_label

PARAMETERS:

Name	Description	Range	Type Allowed
timer_flag#	Timer flag number.	72 to 79	IE, IC
address_label	Branch address.		A

DESCRIPTION:

Branches to the specified address if the timer flag indicated is on.

If the timer flag is off, branching does not occur and execution continues with the next instruction.

RETURNS:

None.

incr_com

SYNTAX:

label incr_com controller#,bits,interrupts

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	controller id#	1 or 2	IE, IC
bits	The incremental value, in bits, to be added to the commanded position.		IE, IC, IV
interrupts	The number of motor interrupts to move the distance specified in 'bits'. Motor interrupts occur every 1 ms.		IE, IC, IV

DESCRIPTION:

This instruction calculates the number of bits to be added to the commanded position, each motor interrupt, as a result of the calculations 'bits/motor interrupts'. Resulting accel/decel rates are not limited. 'bits' may be signed for direction.

RETURNS:

None.

index

SYNTAX:

label index controller#,distance

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	controller id#	1 or 2	IE, IC
distance	Incremental number of bits to move.	(-524288*4096) to (+524288*4096)-1	IE, IC, IV

DESCRIPTION:

Commands the specified controller to index the motor the distance given. The speed of the move is determined by the previously set accel/decel rate and speed.

The resolution is 4096 bits per turn. To index 1 turn, distance would be 4096.

The axis controller status flags **MOTOR BUSY** and **INDEXING** will be set (ON) while the motor is indexing.

This instruction will be ignored if the controller has not received a **drive_on** instruction or if the controller is busy executing a motion instruction. If the instruction cannot be executed, the axis status flag **COMMAND INVALID IN THIS STATE** will be set. This flag will automatically be cleared by the controller at the next communication between the main processor and the axis controller.

RETURNS:

None.

input

SYNTAX:

label input label,length,decimals,value,user_flag

PARAMETERS:

Name	Description	Range	Type Allowed
label	ASCII string (prompt) to be displayed		T
length	Maximum length of input value.	0 to 12	IE, IC
decimals	Decimal places in the input value.	0 to 6	IE, IC
value	The value entered.		IV
user_flag	Flag indicating input is complete.	208 to 255	IE, IC

DESCRIPTION:

Prepares and reads numeric (only) information from the selected port. The port is selected by using the **port_set** instruction.

Input string 'length' determines the number of characters accepted during input. The number of characters includes the sign and decimal point (if applicable) as well as the number of characters.

'Decimal places' indicates the number of values to the right of the decimal point.

Input 'value' is the destination address of the numeric input.

The **port_set** instruction must be executed prior to invoking the **input** instruction.

The characters allowed while entering strings with length exceeding 1 character are +.-1234567890.

If the value being entered has an actual character length greater than the specified input string 'length', the excess characters are ignored. Actual string lengths less than the input string length are permitted and are justified accordingly.

EXAMPLE

If 2.2 is entered
and decimals = 1
then value = 2.2 * (10 ^ 1) = 2.2 * 10 = 22

RETURNS:

The return 'value' contains the numeric conversion of the ASCII string entered at the terminal. This number value is value * (10 ^ decimals).

integer

SYNTAX:

label integer

PARAMETERS:

None.

DESCRIPTION:

This statement assigns a 32 bit storage location, or value, to the name 'label'. The storage location will be initialized to zero when the program is loaded into the Controller.

This instruction requires a 'label'.

RETURNS:

None.

jog_ccw

SYNTAX:

label jog_ccw controller#

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	controller id#	1 or 2	IE, IC

DESCRIPTION:

Commands the specified controller to turn the motor shaft in a counter-clockwise direction using the last set accel/decel rate and speed. Motion will continue until an **f_decel** instruction is executed.

This instruction sets the controller status flags **BUSY** and **JOGGING**. These flags remain set until an **f_decel** instruction causes the motor to reach zero speed.

This instruction will be ignored if the controller has not received a **drive_on** instruction, or if the controller is busy executing a motion instruction. If the instruction cannot be executed, the axis status flag **COMMAND INVALID IN THIS STATE** will be set. This flag will be cleared by a **f_decel** command or a **drive_off** command.

RETURNS:

None.

jog_cw

SYNTAX:

label jog_cw controller#

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	controller id#	1 or 2	IE, IC

DESCRIPTION:

Commands the specified controller to turn the motor shaft in a clockwise direction using the last set accel/decel rate and speed. Motion will continue until an **f_decel** instruction is executed.

This instruction sets the controller status flags **BUSY** and **JOGGING**. These flags remain set until an **f_decel** instruction causes the motor to reach zero speed.

This instruction will be ignored if the controller has not received a **drive_on** instruction, or if the controller is busy executing a motion instruction. If the instruction cannot be executed, the axis status flag **COMMAND INVALID IN THIS STATE** will be set. This flag will cleared by a **f_decel** command or a **drive_off** command.

RETURNS:

None.

l_track_spd

SYNTAX:

label l_track_spd controller#,speed

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	controller id#	1 or 2	IE, IC
speed	Resulting Speed: -128 to +127 RPM	-32768 to +32767	IE, IC, IV

DESCRIPTION:

The specified controller tracks (changes to) the speed indicated divided by 256.

The speed may be changed at any time.

This instruction sets the controller status flags **BUSY** and **JOGGING**. These flags remain set until a **f_decel** instruction causes the motor to reach zero speed.

This instruction will be ignored if the controller has not received a **drive_on** instruction, or if the controller is busy executing a motion instruction. If the instruction cannot be executed, the axis status flag **COMMAND INVALID IN THIS STATE** will be set. This flag will automatically be cleared by the controller at the next communication between the main processor and the axis processor.

RETURNS:

None.

let

SYNTAX:

label let value=operand1 opcode operand2

label let value=function(operand1)

PARAMETERS:

Name	Description	Type Allowed
value	Resulting value.	IV, FV
operand1	Value or constant.	IE, IC, IV, FE, FC, FV
opcode	Operation to be performed.	

opcode	operation
+	addition
-	subtraction
*	multiplication
/	division
&	bitwise and
|	bitwise or
^	bitwise exclusive or
[ ]	array indexing
>>	shift right
<<	shift left
}}	rotate right
{{	rotate left

operand2	Value or constant.	IE, IC, IV, FE, FC, FV
function	Allowable function type.	

function	operation
sqr	square root of operand 1
abs	absolute value of operand 1
neg	negate operand 1

DESCRIPTION:

The only arithmetic operation allowed, using arrays, is a simple assign. Operations such as add, subtract, multiply etc. are not allowed.

The **let** instruction can be used to transfer data (in 4-byte blocks) between program data memory and the volatile memory of an axis controller. The following example will dimension a 100

element array called 'test_array' on controller# 1. The value 1234 will be placed in element 19 (array subscripts are zero based) of this array.

EXAMPLE 1

```
test_array      dim     1,100
                .
                .
                .
                let     test_array[18]=1234
```

The **let** instruction can be used to transfer data (in 4-byte blocks) from the volatile memory of an axis controller to program data memory. The following example will dimension a 100 element array called 'test_array' on controller# 1. The current value of element 3 (array subscripts are zero based) will be placed into value 'x'.

EXAMPLE 2

```
test_array      dim     1,100
                .
                .
                .
                let     x=test_array[2]
```

RETURNS:
None.

let_byte

SYNTAX:

label let_byte destination=source

PARAMETERS:

Name	Description	Type Allowed
destination	Value or cam table element.	IE, IC, IV
source	Value or cam table element.	IE, IC, IV

DESCRIPTION:

Typically used to store and retrieve cam elements, one byte at a time. The only arithmetic operation allowed using the **let_byte** instruction is a simple assign.

Can also be used to pack or split 32 bits to 8 bits or 8 bits to 32 bits.

The **let_byte** instruction can be used to transfer data (in 1-byte blocks) from program data memory to the volatile memory of an axis controller. The following example will dimension a 100 element (400 byte) array called 'test_array' on controller# 1. The value 117 will be placed in byte 19 (array subscripts are zero based) of this array.

EXAMPLE 1

```
test_array      dim             1,100
                .
                .
                .
                let_byte        test_array[18]=117
```

The **let_byte** instruction can be used to transfer data (in 1-byte blocks) from the volatile memory of the axis controller to program data memory. The following example will dimension a 100 element (400 byte) array called 'test_array' on controller# 1. The current value of byte 3 (array subscripts are zero based) will be placed into value 'x'.

EXAMPLE 2

```
test_array      dim             1,100
                .
                .
                .
                let_byte        x=test_array[2]
```

RETURNS:

None.

lock

SYNTAX:

label lock controller#,lock#

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	controller id#	1	IE, IC
lock #	Lock method (see explanations below)		IE, IC

NOTE:

Lock modes 0, 5, 8 and 9 (described below) may be used in conjunction with the switch cam feature. See the **switch_cam** instruction for additional information and examples on the use of the combined features.

DESCRIPTIONS:

Locks the specified controller onto the master position vector as defined by the master angle configuration.

Lock #	Lock Method
0	**cam lock**

The DeltaMax controllers provide a mode of operation that effectively emulates mechanical cams. In the electronic cam mode, the slave axis follows a digital cam based on the master angle from one of the master angle buses. Electronic cams are tables of incremental motor moves. Each incremental value occupies one byte of memory and has an allowable range of -127 to +127. The slave controller uses the current master position to determine the proper index into the electronic cam array. As the index moves through the cam table, the incremental distances are added together to form the proper slave axis position. Linear interpolation is performed between elements in the cam array.

When lock method 0 is executed, the cam executes at the first element in the array. When the end of the cam table is reached, the process begins again at the beginning of the table. This process is bi-directional, with the cam moving to increasing positive number elements with clockwise master rotation and decreasing number elements with counterclockwise rotation.

Further information about cam lock can be found by reviewing the following instructions; cam_data, set_cam_ptr, get_cam_ptr, switch_cam, get_cam_strt, get_cam_end, get_cam_sum, calc_unit_cam, begin_cam, end_cam.

Lock #	Lock Method
1	**simple lock with accel/decel limits**

Lock method 1 provides the electronic equivalent of a gearbox. The electronic gearbox or ratio is driven by the master angle from one of the master angle buses. The master angle is processed by the previously specified **ratio** instruction and an offset is added, resulting in an effective master used to drive the slave command position.

When lock method 1 is executed, the axis controller calculates the instantaneous offset between the master angle processed by the ratio and the slave command position. Once the offset is calculated, master slave lock is accomplished. Once per millisecond thereafter, the slave axis position is updated based on the new master position. The slave motion is limited by the previously set accel/decel rate (see **set_ac_dc**) and is limited to a speed of 5800 RPM. The limiting of the slave acceleration can be an advantage in systems that have rough or rapidly changing master speeds. The limited acceleration on the slave dampens the slave motor and smooths out the operation.

The **ratio** instruction can be executed while in lock method 1. When the **ratio** instruction is executed, the slave controller switches to simply slewing at the **set_ac_dc** rate until the slave reaches the new slave speed. When the speed is matched, a new offset is calculated and lock is resumed. It is important to note that after a **ratio** instruction is executed, a new offset is used. The axis status flag **JOGGING** is on while the slave motor is changing from one speed to another. This flag goes off when the ratio lock equation starts executing. The acceleration rate may be changed during the slew from one speed to another caused by a ratio change. A **set_ac_dc** executed prior to the **ratio** instruction will cause the slew acceleration rate to change.

The result of executing a **lock** command may vary, depending on the value used in the **ratio** command and the command sequence. This can best be illustrated using the following examples:

EXAMPLE 1

```
label       ratio       slave,ratio_value
            .
            .
            .
            lock        slave,1
```

At the moment the **lock** is executed:

a) the slave axis will calculate an internal master/slave "offset", by comparing the master angle with the slave position
b) the slave axis will accelerate at the rate set using the last **set_ac_dc** command
c) the slave axis will "make up" the distance lost during the acceleration time, by running at a faster rate for that same period of time (the slave axis would appear to momentarily "overshoot" the requested ratio)
d) the slave axis will run at the requested ratio while maintaining the internal master/slave "offset"

Subsequent changes in 'ratio_value' will not result in the slave axis "making up" any lost distance while adjusting to a new ratio.

EXAMPLE 2

In this example, the **ratio** command is executed twice. The result of locking the slave axis will be different than that of EXAMPLE 1.

```
label       ratio       slave,0
            .
            .
            .
            lock        slave,1
            .
            .
            ratio       slave,ratio_value
```

At the moment the **lock** is executed:

a) the slave axis will calculate an internal master/slave "offset", but since the ratio is initially set to zero, no motion occurs

At the moment the second **ratio** instruction is executed:

a) the slave axis will accelerate at the rate set using the last **set_ac_dc** command
b) the slave axis will run to the requested ratio while maintaining the internal master/slave "offset" (the slave axis will NOT attempt to "make up" any lost distance while adjusting to the new ratio)

Lock #	**Lock Method**
2	**velocity lock**

In velocity lock, the slave axis tracks the master velocity with accel/decel limit.

Lock #	**Lock Method**
3	**piecewise profile lock**

In piecewise profile lock, the slave axis will execute the piecewise profile in memory when the master angle (modulo 4096) crosses the specified angle from either direction (CW or CCW). This master angle is specified by the instruction **set_trig_pw**. The instruction **prep_profile** must be executed before the **lock** instruction.

Lock #	**Lock Method**
4	**simple lock without accel/decel limits**

Lock method 4 is identical to lock method 1. The only difference is that the slave axis is not limited by the specified slave acceleration rate. This means that during lock, the slave is commanded directly by the effective master position. Note: Any perturbations or roughness in the master will be passed onto the slave with no accel/decel rate limiting.

The **ratio** instruction can be executed during lock method 4. When a new ratio is executed, the slave controller breaks lock, slews to new lock speed at the specified acceleration rate and relocks using the above equation. The **JOGGING** status flag is on during the slew period. Note that after the slew period a new offset is calculated. The acceleration rate may be changed during slew executing a **set_ac_dc** instruction prior to the ratio instruction.

Lock #	Lock Method
5	**cam lock**

Lock method 5 is identical to lock method 0 except that the user has the option of positioning the cam pointer to a position other than the beginning of the cam. This is accomplished by executing the **set_cam_ptr** instruction. Execution of the cam will begin at the position in the cam data table at the cam pointer.

Lock #	Lock Method
6	**keyway lock**

Lock method 6 allows the user to align the absolute position of the slave with the absolute position of the master. The ratio in Lock Method 6 is fixed at 1:1 and may not be changed. The slave controller acceleration rate is limited by the previously executed **set_ac_dc** instruction. The slave top speed is also limited to 5800 RPM.

When the lock instruction is executed, the slave controller executes the following equation every 1 millisecond:

master angle = slave command position

This equation is executed modulo 1 turn (4096 bits) of the master. This means that when the **lock** instruction is executed, the slave will move, at the acceleration rate specified, in the shortest direction to bring the master and slave absolute angles into alignment. Note that Lock Method 6 can cause movement when lock is executed even if the master is at rest.

The slave accel and speed are limited. Hunting can occur if the master rate of change is greater than the specified slave acceleration rate. The rule of thumb of having the slave acceleration rate at least 5 times the expected master rate of change is applicable. The limited acceleration rate can be used to smooth a master with fast perturbations (roughness).

Lock #	Lock Method
7	**undefined**

Lock #	Lock Method
8	**lock cam at specified master angle**

In lock type 8, the slave axis will begin executing its cam when the master axis crosses the specified angle (modulo 4096) from either direction (CW or CCW). This angle is specified using the instruction **set_trig_cam**. It is possible to start execution of the cam from any point in the cam by using the instruction **set_cam_ptr**. The axis status flag **SWITCH CAM PENDING** will be set until the master crosses the specified lock angle and execution of the cam begins.

Lock #	**Lock Method**
9	**lock cam at specified master angle for 1 cycle only**

Lock type 9 is identical to lock type 8 except that execution of the cam will automatically terminate at a rollover in the cam.

RETURNS:

None.

master_scale

SYNTAX:

label master_scale controller#,mscale

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	controller id #	1	IE, IC
mscale	cam related master scale	0 to 12	IE, IC, IV

DESCRIPTION:

The **cam_data** instruction provides cam setup information to the controller. This includes the location of the data which defines the cam, the master scale factor and the data scale factor.

Due to the amount of information it provides, the **cam_data** instruction is a relatively lengthy instruction to execute, compared to most instructions. The **master_scale** instruction can be used when the only information to be changed regarding cam processing is the master scale.

The master scale factor determines the rate at which cam elements are executed relative to the change in master position.

Master Scale Factor	Cam Array Advances Every
12	4096 master bits
11	2048 master bits
10	1024 master bits
9	512 master bits
8	256 master bits
7	128 master bits
6	64 master bits
5	32 master bits
4	16 master bits
3	8 master bits
2	4 master bits
1	2 master bits
0	1 master bits

The **master_scale** instruction can be used to initiate a new master scale factor (cam element execution rate) without the need to pass additional information to the controller in the form of the **cam_data** instruction.

RETURNS:

None.

mod_index

SYNTAX:

label mod_index controller#,speed,rate,distance,degree

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	controller id #	1 or 2	IE, IC
speed	Desired speed in RPM.	1 to 5800 RPM	IE, IC, IV
rate	Desired accel/decel rate in revs/sec/sec.	1 to 1600	IE, IC, IV
distance	Incremental number of bits to move.	(-524288*4096) to(+524288*4096)-1	IE, IC, IV
degree	The relative amount of motion modification.	0 to 9	IE, IC, IV

DESCRIPTION:

The **mod_index** instruction commands the controller to index (incremental motion) the motor the specified distance as defined by the speed, rate and degree provided.

The **mod_index** instruction is similar to the **index** instruction. There are, however, two key differences:

1 - The **mod_index** instruction requires the speed, rate and distance parameters. The **index** instruction requires only the distance parameter. The speed and rate parameters must be preset using the **set_speed** and **set_ac_dc** instructions.

2 - The **mod_index** instruction will initiate a modified sine shape motion that is determined by the degree parameter. The **index** instruction will initiate a normal trapezoidal shape motion with no modification.

A degree entry of ‘0’ indicates no motion modification. The larger the degree entry, the greater the modification to the motion shape.

Resolution is 4096 bits per turn. To index 1 turn, distance would be 4096.

The axis controller status flags MOTOR BUSY and INDEXING will be set (ON) while the motor is indexing.

This instruction will be ignored if the controller has not received a **drive_on** instruction or if the controller is busy executing a motion instruction. If the instruction cannot be executed, the axis

status flag COMMAND INVALID IN THIS STATE will be set. This flag will automatically be cleared by the next axis controller instruction is executed.

RETURNS:

None.

mod_position

SYNTAX:

label	mod_position	controller#,speed,rate,abs_position,degree

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	controller id #	1 or 2	IE, IC
speed	Desired speed in RPM.	1 to 5800 RPM	IE, IC, IV
rate	Desired accel/decel rate in revs/sec/sec.	1 to 1600	IE, IC, IV
abs_position	The absolute position relative to position zero.	(-524288*4096) to(+524288*4096)-1	IE, IC, IV
degree	The relative amount of motion modification.	0 to 9	IE, IC, IV

DESCRIPTION:

The **mod_position** instruction commands the controller to position the motor to the specified abs_position as defined by the speed, rate and degree provided.

The **mod_position** instruction is similar to the **position** instruction. There are, however, two key differences:

1 - The **mod_position** instruction requires the speed, rate and distance parameters. The **position** instruction requires only the abs_position parameter. The speed and rate parameters must be preset using the **set_speed** and **set_ac_dc** instructions.

2 - The **mod_position** instruction will initiate a modified sine shape motion that is determined by the degree parameter. The **position** instruction will initiate a normal trapezoidal shape motion with no modification.

A degree entry of ‘0’ indicates no motion modification. The larger the degree entry, the greater the modification to the motion shape.

Resolution is 4096 bits per turn. To index 1 turn, distance would be 4096.

The axis controller status flags MOTOR BUSY and INDEXING will be set (ON) while the motor is positioning.

This instruction will be ignored if the controller has not received a **drive_on** instruction or if the controller is busy executing a motion instruction. If the instruction cannot be executed, the axis

status flag COMMAND INVALID IN THIS STATE will be set. This flag will automatically be cleared by the next axis controller instruction is executed.

RETURNS:

None.

no_op

SYNTAX:

label no_op

PARAMETERS:

None.

DESCRIPTION:

Performs a 'no operation' instruction. Commonly used to provide a short time delay.

RETURNS:

None.

offset_master

SYNTAX:

label offset_master controller#,offset

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	controller#	1 or 2	IE, IC
offset	Offset to be added to the master position. This value is a signed 32-bit number.		IE, IC, IV

DESCRIPTION:

This command instructs the operating system to add the specified offset value (in bits) to the actual transducer angle of the specified master controller before it is placed on the master angle bus. The offset is an absolute value added to the master angle and is unaffected. Only the data being placed on the master angle bus is affected.

RETURNS:

None.

over_draw

SYNTAX:

label over_draw controller#,speed,limit,distance

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	controller id#	1	IE, IC
speed	Desired speed to start overdraw in RPM. Must not exceed speed of associated index or position instruction.		IE,IC,IV
limit	Desired limit in bits. Represents maximum travel allowed if hardware interrupt does NOT occur.		IE, IC, IV
distance	Distance to travel AFTER the hardware interrupt occurs.		IE, IC, IV

NOTE:

This instruction can only be used in conjunction with hardware interrupts.

DESCRIPTION:

This instruction is typically used in feed to sensor types of applications. Information provided by the **over_draw** instruction modifies an **index** or **position** instruction. As the **index** or **position** instruction nears completion, motor speed will begin to decrease according to the set accel/decel rate. When the motor speed reaches the value specified in the **over_draw** instruction, deceleration will cease. The motor will run at constant speed until one of two conditions are met:

1. The distance specified in 'limit' is reached. Should this occur, the motor will decelerate to a stop.
2. The Controller's associated input module turns on. Should this occur, the position of the motor shaft is immediately noted. The motor will stop 'distance' bits from the point the input module turns on.

It should be noted that the motor shaft position is trapped immediately by the controller's hardware. However, it may be as long as 1 millisecond before the trapped position is serviced. For this reason, the 'distance' value should allow at least enough distance for deceleration **plus** the distance the motor shaft would turn in 1 millisecond at the 'speed' specified in the **over_draw** instruction.

The controller will set the axis status flag **OVERDRAW FAIL** if in the index or position move, the over_draw speed was not reached.

EXAMPLE

```
busy_1          equ             94          busy flag
hwi_armed_1     equ             91          interrupt armed flag
                .
                drive_on.       1
                set_speed       1,400       400 rpm
                set_ac_dc       1,100       100 rev/sec2

! sensor only active during overdraw section of the profile
                set_ovd_mode    1,1
                .
                .
                .
profile
                enable_hwi
                over_draw       1,100,4096,2000

! wait for index to finish.  If interrupt still armed, sensor not tripped
                index           1,81920

loop            if_stat_on      1,busy_1,loop
                if_stat_on      1,hwi_armed_1,od_fail
                .
                (normal processing)
                .
od_fail
                (handle case where sensor not seen)
```

RETURNS:

None.

port_set

SYNTAX:

label port_set port#,baud,protocol

PARAMETERS:

Name	Description	Range	Type Allowed
port#	Port number.	2	IE, IC
baud	Data transmission rates are listed below, use any one of the following: 110, 300, 600, 1200, 2400, 4800, 9600 or 19200.		IE, IC
protocol	Communications characteristics are listed below.		IE, IC

protocol	Description
0	1 stop bit, no parity, XON/XOFF disabled
1	2 stop bits, no parity, XON/XOFF disabled
2	1 stop bit, even parity, XON/XOFF disabled
3	2 stop bits, even parity, XON/XOFF disabled
4	1 stop bit, odd parity, XON/XOFF disabled
5	2 stop bits, odd parity, XON/XOFF disabled
6	RESERVED
7	RESERVED
8	1 stop bit, no parity, XON/XOFF enabled
9	2 stop bits, no parity, XON/XOFF enabled
10	1 stop bit, even parity, XON/XOFF enabled
11	2 stop bits, even parity, XON/XOFF enabled
12	1 stop bit, odd parity, XON/XOFF enabled
13	2 stop bits, odd parity, XON/XOFF enabled

If no parity, then data word is 8 bits. If even or odd parity, then data word is 7 bits.

DESCRIPTION:

Open and initialize the specified communications port.

The **port_set** instruction may be executed at any time.

Port number 2 is the RS-232C port on the Delta Max.

RETURNS:

None.

position

SYNTAX:

label position controller#,abs_position

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	controller id#	1 or 2	IE, IC
abs_position	The absolute position relative to position zero.	(-524288*4096) to (+524288*4096) -1 bits	IE, IC, IV

DESCRIPTION:

Move the controller to the specified position, relative to zero. The speed of the move is determined by the previously set accel/decel rate and speed.

The Delta Max has a resolution of 4096 bits per turn. To position to turn 1, 'abs_position' would be 4096.

RETURNS:

None.

prep_profile

SYNTAX:

label prep_profile controller#,data_label

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	controller id#	1 or 2	IE, IC
data_label	Label of the data for the profile.		T

DESCRIPTION:

Transmits the desired Piecewise Profile data to the specified controller and prepares the data for execution. While the motion controller is performing its' calculations, the **CALCULATING PIECEWISE PROFILE** controller status flag will be on. When profile calculations or execution have been completed, this controller status flag will be turned off. After calculations are completed, the **get_pstat** instruction may be used to determine whether calculations completed successfully.

RETURNS:

None.

preset

SYNTAX:

label preset controller#,value

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	controller id#	2	IE, IC
value	offset value		IE, IC, IV

DESCRIPTION:

Adds the offset value in bits to the master angle used to drive the PLS.

RETURNS:

None.

print

SYNTAX:

label print text_label

PARAMETERS:

Name	Description	Type Allowed
text_label	Label of a text string.	T

DESCRIPTION:

Print the ASCII character string to the port declared in the previous **port_set** instruction.

The execution of a **port_set** instruction must precede the execution of the **print** instruction.

The output to the port is buffered, that is, the speed characteristics of the port do not affect the execution of the Delta Max.

RETURNS:

None.

print_num

SYNTAX:

label print_num length,decimals,value

PARAMETERS:

Name	Description	Range	Type Allowed
length	Number of character places of 'value' parameter.	1 to 16	IE, IC, IV
decimals	Number of decimal places of 'value' parameter.	0 to 6	IE, IC, IV
value	Numeric value to be converted to ASCII characters and displayed according to the length and decimal parameters.		IE, IC, IV

DESCRIPTION:

Print the output value to the port declared in the previous **port_set** instruction.

The format of the printed number is defined by 'length' which declares the maximum number of character positions allowed to represent the numeric value and by 'decimals' which declares the number of numeric positions to the right of the decimal point.

If the specified length is less than the actual length of the value to be printed, a string of asterisks (*) will be printed instead. If the specified length is greater than the actual length of the value to be printed, the number will be right justified accordingly.

The execution of a **port_set** instruction must precede the execution of the **print_num** instruction.

The output to the port is buffered, that is the speed characteristics of the port do not affect the execution of the Delta Max.

RETURNS:

None.

ratio

SYNTAX:

label ratio controller#,ratio

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	controller id#	1	IE, IC
ratio	Desired electronic ratio (in bits).	-32768 to +32767	IE, IC, IV

DESCRIPTION:

Scale the master/slave position vector being passed by the 'ratio' value for the specified controller.

Combined with the **lock** instruction (simple lock method), an electronic 'gear box' with a value ratio may be emulated.

'ratio' is in bits (i.e. 4096 would give a ratio of 1:1).

RETURNS:

None.

read_offset

SYNTAX:

label read_offset controller#,value

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	controller id#	1	IE, IC
value	Offset value in bits.		IV

DESCRIPTION:

Returns the current offset as defined by the **set_offset** instruction.

RETURNS:

The current offset is returned in the designated 'value'.

restart_at

SYNTAX:

label restart_at address_label

PARAMETERS:

Name	Description	Type Allowed
address_label	Branch address.	A

DESCRIPTION:

Clear all **gosub** return addresses and branch to the specified 'address label'.

RETURNS:

None.

return_sub

SYNTAX:

label return_sub

PARAMETERS:

None.

DESCRIPTION:

Return from a subroutine invoked by a **gosub** instruction.

The Delta Max provides 20 levels of subroutine nesting. The program status **STACK OVERFLOW** will occur if more than 20 levels are used.

If a **return_sub** instruction is encountered without a corresponding **gosub**, a program status of **STACK UNDERFLOW** may occur.

RETURNS:

None.

select

SYNTAX:

label select value

PARAMETERS:

Name	Description	Type Allowed
value	Value to be tested.	IV

DESCRIPTION:

Change program flow based on the value of 'value'. When executed, a program branch occurs to the **case** statement which matches the specified 'value'.

If there is no match, program flow resumes at the instruction following the **default** instruction. The **exit_select** instruction causes a branch to the instruction following the **end_select** instruction.

EXAMPLE

```
select          num
        case    1
        .
        .
        exit_select

        case    2
        .
        .
        exit_select

        default
        .
        .
        exit_select
end_select
```

RETURNS:

None.

set_ac_dc

SYNTAX:

label set_ac_dc controller#,rate

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	controller id#	1 or 2	IE, IC
rate	Desired accel/decel rate in revs/sec/sec.	1 to 1600	IE, IC, IV

DESCRIPTION:

Set the accel/decel rate for the specified controller. All motions for this controller (except cam, ratios and piecewise profiles) will be determined by this rate.

The rate is scaled in revs/sec/sec for all motions (except **vel_cw** and **vel_ccw** instructions) for which the rate scale is (revs/sec/sec) / 256.

The default accel/decel rate is 1 revs/sec/sec. This is also the lowest allowable accel/decel rate. The highest allowable rate is 1600 revs/sec/sec.

RETURNS:

None.

set_bit

SYNTAX:

label set_bit bit#,value

PARAMETERS:

Name	Description	Range	Type Allowed
bit#	Number of the bit within the 4-byte value to be set to on (logic 1).	0 to 31	IE, IC, IV
value	The 4-byte area in memory affected by this instruction.		IV

DESCRIPTION:

The specified bit will be set to on (logic 1). If that bit has been previously set, it will remain set. If that bit was previously cleared (logic 0), it will now be set to on.

This instruction has no effect on the remaining bits of this 4-byte value.

This instruction will be ignored if 'bit#' is outside the range of 0 to 31.

RETURNS:

None.

set_cam_ptr

SYNTAX:

label set_cam_ptr controller#,value

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	controller id#	1	IE, IC
value	The new start position within the cam table.	0 to (cam table length -1)	IE, IC, IV

DESCRIPTION:

Repositions the 'start of cam' pointer in the current cam table array for the controller. This instruction must be used in conjunction with **lock** types 5, 8 and 9 in order to execute the cam from within the table, rather than the start of the table.

RETURNS:

None.

set_dnet

SYNTAX:

label set_dnet flag#

PARAMETERS:

Name	Description	Range	Type Allowed
flag#	DeviceNet flag number to be set.	56 to 71	IE, IC

DESCRIPTION:

Allows setting of a Devicenet flag from 56-71, sets the value of the flag boolean level true or one.

RETURNS:

None.

set_flag

SYNTAX:

label set_flag user_flag#

PARAMETERS:

Name	Description	Range	Type Allowed
user_flag#	Number of the user flag.	208 to 255	IE, IC

DESCRIPTION:

Sets the specified user flag.

RETURNS:

None.

set_gl_ccw

SYNTAX:

label set_gl_ccw controller#

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	controller id#	1 or 2	IE, IC

DESCRIPTION:

Set the absolute 0.0 position to the nearest position transducer zero in the counter-clockwise direction. The current local 0.0 will be cleared.

This instruction may be executed while the motor is in motion. If it is performed while the motor is stopped, no motion will occur.

This instruction is often used in conjunction with a zero revolution switch to establish an absolute zero or home position on multi-turn systems.

RETURNS:

None.

set_gl_cw

SYNTAX:

label set_gl_cw controller#

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	controller id#	1 or 2	IE, IC

DESCRIPTION:

Set the absolute 0.0 position to the nearest position transducer zero in the clockwise direction. The current local 0.0 will be cleared.

This instruction may be executed while the motor is in motion. If it is performed while the motor is stopped, no motion will occur.

This instruction is often used in conjunction with a zero revolution switch to establish an absolute zero or home position on multi-turn systems.

RETURNS:

None.

set_local

SYNTAX:

label set_local controller#

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	controller id#	1 or 2	IE, IC

DESCRIPTION:

Set the present motor position as the absolute 0.0 position. Will override the **set_gl_ccw** and **set_gl_cw** instructions, however, the global 0.0 position is not changed.

Used to establish a 'floating' home position.

RETURNS:

None.

set_map

SYNTAX:

label set_map value

PARAMETERS:

Name	Description	Type Allowed
value	The desired map value.	IE, IC, IV

DESCRIPTION:

'map' is an acronym for Master Angle Passing.

The **set_map** instruction defines the master angle configuration within a system unit. The proper value of 'value' can be determined from Figure 3.21.

Only 1 transmitter per bus can be defined. Improper bus configurations will cause the **get_map_stat** instruction to return a non-zero value.

RETURNS:

None.

set_mcf

SYNTAX:

label set_mcf controller#,value

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	controller id#	2	IE, IC
value	Multi Function Controller configuration value.		IE, IC, IV

DESCRIPTION:

This instruction is used to configure the multi-function controller on the DeltaMax, which is controller #2. This controls the Pseudo Axis, Master Angle Bus Network, Fiber Optic Network and Programmable Limit Switch (PLS) functions.

RETURNS:

None.

set_ms_tmr

SYNTAX:

label set_ms_tmr timer_flag#,time

PARAMETERS:

Name	Description	Range	Type Allowed
timer_flag#	Timer flag number	72 to 79	IE, IC
time	Time to set.	1 to 65535	IE, IC, IV

DESCRIPTION:

Activates or enables ‘timer flag#’ for a time in milliseconds equal to ‘time’.

The specified ‘timer flag#’ remains set until 'time' has expired.

RETURNS:

None.

set_offset

SYNTAX:

label set_offset controller#,value

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	controller id#	1	IE, IC
value	Desired offset in bits	-32768 to +32767	IE, IC, IV

DESCRIPTION:

Offsets all positions by the specified value in bits.

This instruction will cause the motor to move the distance specified by 'value' within the next motor interrupt.

RETURNS:

None.

set_ovd_mode

SYNTAX:

label set_ovd_mode controller#,mode

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	controller id#	1	IE, IC
mode	Selects the section of the motion profile where the overdraw sensor is active.	0 or 1	IE, IC

DESCRIPTION:

Selects the section of the motion profile where the overdraw sensor is active.

Mode 0 - the sensor is active through the entire profile.

Mode 1 - the sensor is only active during the overdraw section of the profile.

In mode 0, when the sensor is activated during the index part of the profile, the index will be treated as a regular index with no search speed section executed.

RETURNS:

None.

set_pls_ang

SYNTAX:

label set_pls_ang controller#,on_angle,off_angle,module#

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	controller id#	2	IE, IC
on_angle	Turn ON module# at this angle in bits.	0 to 4095 bits (representing 360 degrees)	IE, IC, IV
off_angle	Turn OFF module# at this angle in bits.	0 to 4095 bits (representing 360 degrees)	IE, IC, IV
module#	Desired output module.	16 to 31	IE, IC, IV

DESCRIPTION:

PLS is an acronym for Programmable Limit Switch.

This instruction defines the action that will occur on the designated PLS output unit.

'Module#' 16 to 23 are mapped to the PLS output modules. 'Module#' 24 to 31 are mapped internally to flags.

Each time a **set_pls_ang** instruction is issued, the axis controller executes a sorting algorithm and will activate its' **CALCULATION IN PROGRESS** flag. The program must verify that this flag is no longer active before executing subsequent **set_pls_ang** instructions.

The 'rollover point' is the master cycle length in bits. The default 'rollover point' is 4096. That is, the cycle is defined as a value between 0 and 4095. Based on this 4096 bit cycle, a PLS module will be turned ON when it crosses into the ON range and will be turned OFF when it crosses into the OFF range, based on the master source angle.

NOTE: A PLS module must be enabled using the **set_pls_mask** instruction before it can be used.

RETURNS:

None.

set_pls_cnt

SYNTAX:

label	set_pls_cnt	controller#,pls_count

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	controller id #	2	IE, IC
pls_count	The PLS cycle count.	4 to 8,388,607	IE, IC, IV

DESCRIPTION:

The **set_pls_cnt** instruction defines the cycle count for the PLS function. This also determines the range of allowable entries for the ON ANGLE and OFF ANGLE parameters used in the **set_pls_ang** instruction.

For example, the following **set_pls_cnt** instruction is executed:

label	**set_pls_cnt**	**2,16000**

This would establish a PLS cycle of 16000 bits which will range between 0 and 15999. Once the PLS angle 15999 is passed, the cycle rolls over and begins again at 0.

A subsequent **set_pls_ang** instruction must then define their ON ANGLE and OFF ANGLE parameters within the range of 0 to 15999, otherwise, an ILLEGAL ARGUMENT will be detected and flag # 144 will be turned ON.

RETURNS:

None.

set_pls_mask

SYNTAX:

label set_pls_mask controller#,value

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	controller id#	2	IE, IC
value	The currently defined PLS mask value.		IE, IC, IV

DESCRIPTION:

PLS is an acronym for Programmable Limit Switch.

Defines the PLS mask value used to enable/disable pls functioning of the pls outputs. The 2 high order bytes of 'value' represent the 16 associated output modules/flags.

Those bits set in 'value' will be masked with the currently defined PLS state to determine which PLS modules will continue to be affected.

The the instruction **set_pls_mask** is used to control the activity of the output modules without clearing and/or redefining the output modules using the **set_pls_ang** instruction.

EXAMPLE:
Setting bit 16 of 'value' enables PLS #16 and setting bit 31 of value enables PLS #31. Bits 0 to 15 are unused.

RETURNS:

None.

set_speed

SYNTAX:

label set_speed controller#,speed

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	controller id#	1 or 2	IE, IC
speed	Desired speed in RPM.	1 to 5800 RPM	IE, IC, IV

DESCRIPTION:

Set the speed of the specified controller to 'speed'. If no set_speed command is issued the controller will use 1 RPM.

RETURNS:

None.

set_swi_mask

SYNTAX:

label set_swi_mask value

PARAMETERS:

Name	Description	Type Allowed
value	The software interrupt mask value.	IE, IC, IV

DESCRIPTION:

Determines which of the 32 software interrupts are active. A value of 1 in the corresponding bit of the mask 'value' enables the software interrupt.

RETURNS:

None.

set_tmr

SYNTAX:

label set_tmr timer_flag#,time

PARAMETERS:

Name	Description	Range	Type Allowed
timer_flag#	Timer flag number.	72 to 79	IE, IC
time	Time to set.		IE, IC, IV

DESCRIPTION:

Activates or enables a timer for a time = 'time'. Time is given in .01 second intervals.

The specified flag remains set until 'time' has expired.

RETURNS:

None.

set_trig_cam

SYNTAX:

label set_trig_cam controller#,master_angle

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	controller id#	1	IE, IC
master_angle	Angle of the master controller which initiates a trigger.	0 to +4095	IE, IC, IV

DESCRIPTION:

Sets the trigger point for lock methods 8 & 9 (see **lock**). When the master controller angle crosses the trigger angle from either direction, the slave axis begins executing the cam at the specified cam pointer. (The cam pointer will be set to zero if no **set_cam_ptr** instruction has been executed prior to the lock).

NOTE:

A set_trig_cam instruction will override a previous **set_trig_cam** instruction that has not already been implemented.

RETURNS:

None.

set_trig_pw

SYNTAX:

label set_trig_pw controller#,master_angle

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	controller id#	1	IE, IC
master_angle	Angle of the master controller which initiates a trigger.	0 to +4095	IE, IC, IV

DESCRIPTION:

Sets the trigger point for lock method 3 (see **lock**). When the master controller angle equals the angle position, a Piecewise Profile is executed.

RETURNS:

None.

stop_input

SYNTAX

label stop_input

PARAMETERS:

None.

DESCRIPTION:

Terminates all active **input** instructions and clears the input buffer area of all characters.

RETURNS:

None.

swi_if_off

SYNTAX:

label swi_if_off interrupt#,flag,subroutine_label

PARAMETERS:

Name	Description	Range	Type Allowed
interrupt#	Interrupt number.	0 to 31	IE, IC
flag	Flag to trigger the interrupt.	0 to 255	IE, IC
subroutine_label	Subroutine branch address.		A

DESCRIPTION:

Sets up an event which will trigger when 'flag' changes from on to off.

RETURNS:

None.

swi_if_on

SYNTAX:

label swi_if_on interrupt#,flag,subroutine_label

PARAMETERS:

Name	Description	Range	Type Allowed
interrupt#	Interrupt number.	0 to 31	IE, IC
flag	Flag to trigger the interrupt.	0 to 255	IE, IC
subroutine_label	Subroutine branch address.		A

DESCRIPTION:

Sets up an event which will trigger when 'flag' changes from off to on.

RETURNS:

None.

switch_cam

SYNTAX:

label switch_cam controller#,start element,# of elements

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	controller id#	1	IE, IC
start element	Element relative to axis card memory array element zero		IE, IC, IV
# of elements	Number of cam elements		IE, IC, IV

NOTES:

1) The **lock** instruction will only be executed when the **BUSY** flag is off, otherwise, it will be ignored.

2) A **switch_cam** instruction will override a previous **switch_cam** instruction that has not been already implemented.

DESCRIPTION:

The **SWITCH CAM** instruction allows the programmer to switch from executing one cam to another. The programmer specifies the starting element relative to element 0 in the axis controller's 28K array and the number of elements to be executed.

The manner in which the **SWITCH CAM** instruction works, is dependent on a number of parameters:

1) the **lock** type being used

2) the order in which the **switch_cam** and **lock** instructions are executed

3) the current state of the axis controller, as defined by its axis status flags

When a cam is being executed and a **switch_cam** instruction is issued, the **LOCK PENDING** status flag is set until the cam "rolls" from the last to the first or the first to the last cam element. When the "roll" occurs, the **LOCK PENDING** flag is cleared, the **MASTER/SLAVE** lock is set and the new cam begins execution.

Execution of an **unlock** command will clear the **LOCK PENDING** axis status flag, if currently set.

EXAMPLE 1:

When using the **switch_cam** instruction with lock mode 0 or 5 (assume the axis is not busy and a **cam_data** instruction has been executed).

DELTA MAX COMMAND	RESULT
switch_cam	a) sets the start/end of cam pointers b) no affect on axis status flags
lock	a) turn on **MASTER/SLAVE LOCK** and **BUSY** flags

In this example, the switch takes place immediately (affecting the cam pointers) and the actual lock will occur at the moment the **lock** command is issued.

EXAMPLE 2:
When using the **switch_cam** instruction with lock mode 0 or 5 (assume the axis is not busy and a **cam_data** instruction has been executed).

DELTA MAX COMMAND	RESULT
lock	a) turn on **MASTER/SLAVE LOCK** and **BUSY** flags
switch_cam	a) sets the start/end of cam pointers for the next cam to be executed b) turns on the **LOCK PENDING** flag

In this example, a lock takes place immediately. The **switch_cam** instruction will set up the cam pointers to be used for the next cam, that is, the cam to be executed upon completion of the current cam.

EXAMPLE 3:
When using the **switch_cam** instruction with lock mode 8 or 9 (assume the axis is not busy and a **cam_data** instruction has been executed).

DELTA MAX COMMAND	RESULT
set_trig_cam	a) sets the lock trigger angle
switch_cam	a) sets the start/end of cam pointers b) no affect on axis status flags
lock	a) turn on **LOCK PENDING** and **BUSY** flags

In this example, the switch takes place immediately (affecting the cam pointers) and the actual lock will occur at the moment the master angle has crossed the defined trigger angle. Once the lock takes place, the **LOCK PENDING** flag is turned off and the **MASTER/SLAVE LOCK** flag is turned on.

EXAMPLE 4:

When using the **switch_cam** instruction with lock mode 8 or 9 (assume the axis is not busy and a **cam_data** instruction has been executed).

DELTA MAX COMMAND	RESULT
set_trig_cam	a) sets the lock trigger angle
lock	a) turn on **LOCK PENDING** and **BUSY** flags

switch_cam	if the **LOCK PENDING** flag is on, then
	a) sets the start/end of cam pointers
	b) no affect on axis status flags
	c) the actual lock occurs at the trigger angle
	
	if the **MASTER/SLAVE LOCK** flag is on, then
	a) turns on the **LOCK PENDING** flag
	b) the switch occurs upon completion of the current cam

In this example, if the **LOCK PENDING** flag is on at the moment the switch is executed, the result of the switch is that the cam start/end pointers will be realigned and the lock will still occur when the master angle crosses the trigger angle.

In this example, if the **LOCK PENDING** flag is off at the moment the switch is executed (the master angle has already crossed the trigger angle), the current cam will be completed and then the switch will occur.

RETURNS:

None.

sys_fault

SYNTAX:

label sys_fault

PARAMETERS:

None.

DESCRIPTION:

Stop execution of the program and set the system fault bit in the DeltaMax status word.

An **f_decel** instruction is executed for all controllers.

RETURNS:

None.

sys_return

SYNTAX:

label sys_return

PARAMETERS:

None.

DESCRIPTION:

Stop execution of the program and set the system return bit in the DeltaMax status word.

An **f_decel** instruction is executed for all controllers.

RETURNS:

None.

text

SYNTAX:

label text "ASCII string"

PARAMETERS:

Name	Description
string	ASCII string (enclosed in quotes).

DESCRIPTION:

Defines a string of characters for use with the **print** and **input** instructions.

Control and special keyboard characters which cannot be typed may be used by entering the decimal equivalent of the ASCII character(s) enclosed in '<' and '>'.

This instruction requires a 'label'.

RETURNS:

None.

track_spd

SYNTAX:

label track_spd controller#,speed

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	controller id#	1 or 2	IE, IC
speed	Desired speed.	-5800 to +5800 RPM	IE, IC, IV

DESCRIPTION:

The specified controller tracks (changes to) the speed indicated. All speed changes occur at the previously set accel/decel rate. The speed may be changed at any time.

This instruction sets the controller status flags **BUSY** and **JOGGING**. These flags remain set until a **f_decel** instruction causes the motor to reach zero speed.

This instruction will be ignored if the controller has not received a **drive_on** instruction, or if the controller is busy executing a motion instruction. If the instruction cannot be executed, the axis status flag **COMMAND INVALID IN THIS STATE** will be set. This flag will automatically be cleared by the controller at the next communication between the main processor and the axis controller. This occurs automatically every 100 milliseconds even if no program instruction directs communication to occur.

RETURNS:

None.

trap_pos

SYNTAX:

label trap_pos controller#

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	controller id#	1	IE, IC

DESCRIPTION:

When the hardware interrupt signal on the controller is detected, the current position will be saved immediately.

This position can be retrieved later using the **get_trap_pos** instruction.

EXAMPLE

```
hwi_armed_1   equ          91          ! interrupt armed flag
              .
              .
              enable_hwi
              trap_pos     1

! wait for indication that position has been trapped

loop
              if_stat_on   hwi_armed_1,loop
              get_trap_pos 1,pos
              .
```

RETURNS:

None.

turn_off

SYNTAX:

label turn_off I/O flag#

PARAMETERS:

Name	Description	Range	Type Allowed
I/O flag#	Output module number.	16 to 23	IE, IC

DESCRIPTION:

Causes the specified I/O flag to be turned off.

The **turn_off** instruction cannot be used for output flag positions equipped with input modules.

RETURNS:

None.

turn_on

SYNTAX:

label turn_on I/O flag#

PARAMETERS:

Name	Description	Range	Type Allowed
I/O flag#	Output module number.	16 to 23	IE, IC

DESCRIPTION:

Causes the specified I/O flag to be turned on.

The **turn_on** instruction cannot be used for output flag positions equipped with input modules.

RETURNS:

None.

unlock

SYNTAX:

label unlock controller#,mode#

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	controller id#	1	IE, IC
mode#	unlock method	0 or 1	IE, IC

DESCRIPTION:

Terminate the master/slave or CAM lock.

Using 'mode' 0, the controller will decelerate to zero speed at the previously set accel/decel rate.

Using 'mode' 1, the controller will unlock from the master, but will continue to run at the last commanded speed. It will not decelerate until it is commanded to do so using the **f_decel** instruction.

RETURNS:

None

vel_ccw

SYNTAX:

label vel_ccw controller#

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	controller id#	1 or 2	IE, IC

DESCRIPTION:

The specified controller accelerates and runs in a counter-clockwise direction at the previously set accel/decel rate and speed. The current accel/decel rate is divided by 256 before acceleration begins.

Accel/decel rate and speed may be modified while executing the **vel_ccw** instruction.

This instruction sets the controller status flags **BUSY** and **JOGGING**. These flags remain set until a **f_decel** instruction causes the motor to reach zero speed.

This instruction will be ignored if the controller has not received a **drive_on** instruction, or if the controller is busy executing a motion instruction. If the instruction cannot be executed, the axis status flag **COMMAND INVALID IN THIS STATE** will be set. This flag will automatically be cleared by the controller at the next communication between the main processor and the axis controller. This occurs automatically every 100 milliseconds even if no program instruction directs communication to occur.

RETURNS:

None.

vel_cw

SYNTAX:

label vel_cw controller#

PARAMETERS:

Name	Description	Range	Type Allowed
controller#	controller id#	1 or 2	IE, IC

DESCRIPTION:

The specified controller accelerates and runs in a clockwise direction at the previously set accel/decel rate and speed. The current accel/decel rate is divided by 256 before acceleration begins.

Accel/decel rate and speed may be modified while executing the **vel_cw** instruction.

This instruction sets the controller status flags **BUSY** and **JOGGING**. These flags remain set until a **f_decel** instruction causes the motor to reach zero speed.

This instruction will be ignored if the controller has not received a **drive_on** instruction, or if the controller is busy executing a motion instruction. If the instruction cannot be executed, the axis status flag **COMMAND INVALID IN THIS STATE** will be set. This flag will automatically be cleared by the controller at the next communication between the main processor and the axis controller. This occurs automatically every 100 milliseconds even if no program instruction directs communication to occur.

RETURNS:

None.

Analog Functions

```
!-------------------------------------------------------------------------
! TITLE:       Analog Functions
!
! DATE:        12/8/97
!
! DESCRIPTION: This program outputs an analog signal which changes in
!              amplitude from 0 Vdc to +5 Vdc and does so cyclically.
!              The output is verified by hardwiring it to an input and
!              monitoring that input.
!
!              Hardware is wired in the following manner:
!              ch1 = ch 3
!
! EQUIPMENT:   DeltaMax
!
! COMMANDS:    analog_in           analog_out          analog_rt
!              analog_zo           clr_sys_error
!
! The above commands are commonly used in many programs.  This
! illustration provides the context and syntax required to successfuly
! integrate these commands into a complete program.  While the use of
! these commands varies from program to program, the following example
! will give the programmer an indication of a typical usage. An illegal
! argument has been purposefully set in the following example so as to
! make use of the 'clr_sys_error' instruction.
!-------------------------------------------------------------------------

!-------------------------------------------------------------------------
!                                                                        -
! Declare constants and variables                                        -
!                                                                        -
!-------------------------------------------------------------------------

!
! Axis related constants
!

ANALOG          equ             3               ! Analog controller card

CHANNEL_IN      equ             1               ! Analog input channel.
CHANNEL_OUT     equ             3               ! Analog output channel.

!
! I/O and Flags
!

SYS_OK          equ             16              ! System okay (output)

TIMER           equ             72              ! Timer to create time
!                                                 delays.

ERROR           equ            144              ! Flag used when system error
!                                                 is executed.
```

```
!
! Variable definitions
!

rate            integer         ! The slew rate used for the analog
!                                 channels.
change          integer         ! The incremental change for the analog
!                                 output.
alog_out        integer         ! Defines the value in bits of the analog
!                                 output.
alog_in         integer         ! Defines the value in bits of the analog
!                                 input.
diff            integer         ! Stores the difference between two
!                                 variables.
time            integer         ! The time used to create a delay.

!-------------------------------------------------------------------------
!                                                                        -
! Initialize variables                                                   -
!                                                                        -
!-------------------------------------------------------------------------

Defaults        let             rate=1023
                let             change=256
                let             alog_out=0
                let             time=20

!-------------------------------------------------------------------------
!                                                                        -
! Setup a software interrupt when a system error occurs, a flag is set  -
! when an error (illigal argument) occurs.                               -
!                                                                        -
!-------------------------------------------------------------------------
                swi_if_on       0,ERROR,Clr_err  ! go to subroutine 'Clr_err'
                                                 ! when an error occurs
                enable_swi

!-------------------------------------------------------------------------
!                                                                        -
! Program Setup                                                          -
!  Define the analog zero and slew rates for the channels.               -
!                                                                        -
!-------------------------------------------------------------------------

Setup           analog_zo       ANALOG,CHANNEL_IN,0
                gosub           Delay
                analog_rt       ANALOG,CHANNEL_IN,rate
                analog_rt       ANALOG,CHANNEL_OUT,rate
                gosub           Delay

!-------------------------------------------------------------------------
!                                                                        -
! Main body of the program.                                              -
!  Adjust the analog output by an incremental amount.                    -
!  Wait for the output to settle.                                        -
!  Measure the analog input and compare the values.                      -
```

```
!  If out of range then stop system, otherwise continue in Loop.        -
!                                                                       -
!-----------------------------------------------------------------------

Main            turn_on         SYS_OK

Loop            let             alog_out=alog_out+change
                if              alog_out<=1024,Adjust_Out  ! '=' creates an
                                                           ! illegal argument
                let             alog_out=0

Adjust_Out      analog_out      ANALOG,CHANNEL_OUT,alog_out
                gosub           Delay

                analog_in       ANALOG,CHANNEL_IN,alog_in
                let             diff=alog_out-alog_in
                let             diff=abs(diff)
                if              diff<50,Loop
                turn_off        SYS_OK
                sys_fault

!-----------------------------------------------------------------------
!                                                                       -
! Subroutine used to create a time delay.                               -
!  Wait until the timer times out.                                      -
!                                                                       -
!-----------------------------------------------------------------------
Delay           set_tmr         TIMER,time
Wait_Time       if_tmr_on       TIMER,Wait_Time
                return_sub

!-----------------------------------------------------------------------
!                                                                       -
! Subroutine used to clear an illegal argument.A timer is used so that  -
! the error is displayed on the seven segment display,for approx. 2secs.-
!                                                                       -
!-----------------------------------------------------------------------
Clr_err         set_tmr         73,200
Wait_Time1      if_tmr_on       73,Wait_Time1

                clr_sys_error   ERROR

                set_tmr         73,200
Wait_Time2      if_tmr_on       73,Wait_Time2

                enable_swi
                return_sub
```

CAMS with switch_cam

```
!-------------------------------------------------------------------------
! TITLE:       CAMS with switch_cam
!
! DATE:        12/3/97
!
! DESCRIPTION: This program uses two distinct CAMs to illustrate
!              the use of a 'switch_cam' and other associated cam
!              functions.  Software interrupts are used to demonstrate
!              the ability to change cams on the fly, as well as
!              the ability to reposition and lock a cam at a
!              different point.
!
! EQUIPMENT:   DeltaMax
!
! COMMANDS: begin_cam          cam               cam_data
!           clr_flag           end_cam           f_decel
!           get_cam_cnt        get_cam_end       get_cam_ptr
!           get_cam_strt       get_map           get_map_stat
!           get_mcf            get mcf stat      if_flag_off
!           if_flag_on         if_io_off         if_stat_on
!           lock               restart_at        set_cam_ptr
!           set_flag           set_map           set_mcf
!           switch_cam         sys_fault         sys_return
!           turn_off           turn_on           unlock
!
! The above commands are commonly used in many programs.  This
! illustration provides the context and syntax required to successfuly
! integrate these commands into a complete program.  While the use of
! these commands varies from program to program, the following example
! will give the programmer an indication of a typical usage.
!-------------------------------------------------------------------------

              declare       on

!-------------------------------------------------------------------------
!                                                                        -
! Declare constants and variables                                        -
!                                                                        -
!-------------------------------------------------------------------------

!
!  Axis related constants
!

PEN           equ           1

PSEUDO        equ           2

BUSY          equ           78               ! busy flag offset
DOWN          equ           77               ! down flag offset
PEN_BUSY      equ           (PEN*16)+BUSY    ! busy flag - PEN
PSEUDO_BUSY   equ           (PSEUDO*16)+BUSY ! busy flag - PSEUDO
PEN_DOWN      equ           (PEN*16)+DOWN    ! down flag - PEN
PEN_SWITCH    equ           81               ! lock pending - PEN
```

```
UNLK_MODE_0     equ             0               ! unlock mode 0

LK_MODE_0       equ             0               ! lock at start of cam
LK_MODE_5       equ             5               ! lock within the cam
MS              equ             6               ! 64 master bits/cam element
DS              equ             0               ! cam el = cam el * 2^0
PSEUDO_MABA     equ             10              ! PSEUDO sends data to
!                                                 MABA
MABA_TALKER     equ             30              ! PSEUDO is talker on MABA

MABA_LISTEN     equ             23              ! PEN is listener on MABA
END_CAM         equ             128             ! Value for end of cam

!
!  I/O and Flags
!

SWITCH_RQST     equ             0               ! Request to switch cam (input)
REPEAT          equ             1               ! Repeat section of cam
!                                                 (input)
RESET           equ             6               ! Reset a PEN fault (input)
SYS_OK          equ             16              ! System okay (output)


TIMER           equ             72              ! Timer for fixed delay.

IN_CAM_FWD      equ             254             ! Flag to indicate current
!                                                 cam.
SWITCH          equ             255             ! Flag to indicate a switch
!                                                 request.

!
!  Variable definitions
!

speed           integer         ! Defines the speed of the PSEUDO.
length          integer         ! Determines the length of the cam.

start           integer         ! Starting location of cam.
cam_start       integer         ! Return variable for cam starting
!                                 location.
cam_end         integer         ! Return variable for cam ending
!                                 location.
number_el       integer         ! Number of cam elements.
count           integer         ! Return variable for number of cam
!                                 cycles.
pointer         integer         ! Return variable for the cam pointer.

new_pointer     integer         ! The variable used to reposition the
!                                 cam pointer.
last_el         integer         ! Defines the location of the last
!                                 cam element.
temp            integer         ! Used as a temporary storage variable.
ctr             integer         ! Used as a loop counter.
ctr2            integer         ! Used as a loop counter.
time            integer         ! Determines the delay time for a
```

```
!                                 timer.
time2           integer         ! Determines the delay time for a
!                                 timer.
map             integer         ! Used to configure the MASTER/SLAVE on
!                                 the MAB.
map_back        integer         ! Return variable for the current 'map'
!                                 setup.
map_stat        integer         ! map_stat = 0 if map OK, <> 0 if error.
mcf             integer         ! Used to configure the PSEUDO.

mcf_stat        integer         ! mcf_stat = 0 if map OK, <> 0 if error.
mcf_back        integer         ! Return variable for the current 'mcf'
!                                 setup.
!
!  Cam tables and data arrays
!

pen_cams        dim             PEN,200

forward_cam     begin_cam
                cam             64,64,64,64,64,64,64,64
                cam             64,64,64,64,64,64,64,64
                cam             64,64,64,64,64,64,64,64

                cam             64,64,64,64,64,64,64,64
                cam             64,64,64,64,64,64,64,64
                cam             64,64,64,64,64,64,64,64
                cam             64,64,64,64,64,64,64,64
                cam             64,64,64,64,64,64,64,64
                cam             64,64,64,64,64,64,64,64
                cam             64,64,64,64,64,64,64,64
                cam             64,64,64,64,64,64,64,64
                cam             64,64,64,64,64,64,64,64

                cam             64,64,64,64,64,64,64,64
                cam             64,64,64,64,64,64,64,64
                cam             64,64,64,64,64,64,64,64
                cam             64,64,64,64,64,64,64,64
                cam             -32,-32,-32,-32,-32,-32,-32,-32
                cam             -32,-32,-32,-32,-32,-32,-32,-32
                cam             -32,-32,-32,-32,-32,-32,-32,-32
                cam             -32,-32,-32,-32,-32,-32,-32,-32
                cam             -32,-32,-32,-32,-32,-32,-32,-32

                cam             -32,-32,-32,-32,-32,-32,-32,-32
                cam             -32,-32,-32,-32,-32,-32,-32,-32
                cam             -32,-32,-32,-32,-32,-32,-32,-32
                cam             0,0,0,0,0,0,0,0
                cam             0,0,0,0,0,0,0,0
                cam             0,0,0,0,0,0,0,0
                cam             0,0,0,0,0,0,0,0
                end_cam

reverse_cam     begin_cam
                cam             -32,-32,-32,-32,-32,-32,-32,-32
```

```
                cam             -32,-32,-32,-32,-32,-32,-32,-32
                cam             -32,-32,-32,-32,-32,-32,-32,-32
                cam             -32,-32,-32,-32,-32,-32,-32,-32
                cam             -32,-32,-32,-32,-32,-32,-32,-32
                cam             -32,-32,-32,-32,-32,-32,-32,-32
                cam             -32,-32,-32,-32,-32,-32,-32,-32
                cam             -32,-32,-32,-32,-32,-32,-32,-32
                cam             -32,-32,-32,-32,-32,-32,-32,-32

                cam             -32,-32,-32,-32,-32,-32,-32,-32
                cam             -32,-32,-32,-32,-32,-32,-32,-32
                cam             -32,-32,-32,-32,-32,-32,-32,-32
                cam             -32,-32,-32,-32,-32,-32,-32,-32
                cam             -32,-32,-32,-32,-32,-32,-32,-32
                cam             -32,-32,-32,-32,-32,-32,-32,-32
                cam             -32,-32,-32,-32,-32,-32,-32,-32
                cam             64,64,64,64,64,64,64,64

                cam             64,64,64,64,64,64,64,64
                cam             64,64,64,64,64,64,64,64
                cam             64,64,64,64,64,64,64,64
                cam             64,64,64,64,64,64,64,64
                cam             64,64,64,64,64,64,64,64
                cam             64,64,64,64,64,64,64,64
                cam             64,64,64,64,64,64,64,64
                cam             0,0,0,0,0,0,0,0
                cam             0,0,0,0,0,0,0,0

                cam             0,0,0,0,0,0,0,0
                cam             0,0,0,0,0,0,0,0
                end_cam

!-------------------------------------------------------------------------
!                                                                        -
! Initialize variables                                                   -
!                                                                        -
!-------------------------------------------------------------------------


Defaults        let             speed=100
                let             time=1
                let             time2=50

!-------------------------------------------------------------------------
!                                                                        -
! Program Setup                                                          -
!  Define axis motion parameters.                                        -
!  Configure map.                                                        -

!  Load cam tables into pen_cams array.                                  -
!  Define software interrupts.                                           -
!                                                                        -
!-------------------------------------------------------------------------

Setup           set_ac_dc       PSEUDO,100
```

```
               set_ac_dc      PEN,400

               gosub          Set_Config
               gosub          Load_Cams

               swi_if_on      0,PEN_DOWN,Fault

               swi_if_on      1,SWITCH_RQST,Make_Switch
               swi_if_on      2,REPEAT,Reposition_Cam
               enable_swi

!-------------------------------------------------------------------------
!                                                                        -
! Main body of program                                                   -
!  Enable PEN.                                                           -
!  Start motion.                                                         -

!  Check conditons for a 'switch_cam'.                                   -
!    If switch is not correct then stop system.                          -
!  Get information about current cam.                                    -
!  Remain in Loop, waiting for software interrupts.                      -
!                                                                        -
!-------------------------------------------------------------------------

Main           turn_on        SYS_OK

               drive_on       PEN


               lock           PEN,LK_MODE_0
               track_spd      PSEUDO,speed

Loop           if_flag_off    SWITCH,Continue
               if_stat_on     PEN_SWITCH,Continue
               get_cam_strt   PEN,cam_start
               get_cam_end    PEN,cam_end
               let            number_el=cam_end-cam_start
               let            number_el=number_el+1
               if             number_el<>length,System_Halt
               clr_flag       SWITCH


Continue       get_cam_ptr    PEN,pointer
               get_cam_cnt    PEN,count
               set_tmr        TIMER,time
Delay          if_tmr_on      TIMER,Delay
               goto           Loop

!-------------------------------------------------------------------------
!                                                                        -
! Software Interrupt Routine indicating a request to switch cams.       -
!  Test for current cam and switch to alternate cam.                     -

!  Set the flag that indicates a cam switch has occurred.                -
!  Enable the software interrupts and return.                            -
!                                                                        -
!-------------------------------------------------------------------------
```

```
Make_Switch    if_flag_on     IN_CAM_FWD,Switch_Reverse
               set_flag       IN_CAM_FWD
               let            start=0
               let            length=208

               switch_cam     PEN,start,length
               goto           Exit_Switch

Switch_Reverse no_op
               clr_flag       IN_CAM_FWD
               let            start=208
               let            length=208
               switch_cam     PEN,start,length

Exit_Switch    set_flag       SWITCH
               enable_swi
               return_sub

!-------------------------------------------------------------------------
!                                                                        -

! Software Interrupt Routine used to repeat a section of the current cam.-
!  Stop the PEN.                                                         -
!  Wait for a short period of time.                                      -
!  Reposition the cam pointer then start again.                          -
!  Enable the software interrupts and return.                            -
!                                                                        -
!-------------------------------------------------------------------------


Reposition_Cam no_op
               unlock         PEN,UNLK_MODE_0
Wait_Busy      if_stat_on     PEN_BUSY,Wait_Busy

               set_tmr        TIMER,time2
Wait_Time      if_tmr_on      TIMER,Wait_Time

               let            new_pointer=128
               if             pointer<128,Cont_Repos
               let            new_pointer=0

Cont_Repos     set_cam_ptr    PEN,new_pointer
               lock           PEN,LK_MODE_5
               enable_swi

               return_sub

!-------------------------------------------------------------------------
!                                                                       -
! Software Interrupt Routine used for a PEN fault condition.            -
!  Turn off the System Ok output.                                       -
!  Check to see that all motion has stopped.                            -
!  Wait for reset input, then restart at Main.                          -

!                                                                       -
!-------------------------------------------------------------------------
```

```
Fault           no_op
                turn_off        SYS_OK
                f_decel         PEN
                f_decel         PSEUDO
Wait_Busy1      if_stat_on      PEN_DOWN,Wait_Busy2
                if_stat_on      PSEUDO_BUSY,Wait_Busy1

Wait_Busy2      if_stat_on      PSEUDO_BUSY,Wait_Busy2

Start_Over      if_io_off       RESET,Start_Over

                drive_off       PEN
                enable_swi
                restart_at      Main

!-------------------------------------------------------------------------
!                                                                        -
! Subroutine used when an incorrect switch occurs.                       -
!  Turn off the System Ok output.                                        -
!  Check to see that all motion has stopped.                             -

!  Disable PEN drive, and indicate a system fault on the System Status. -
!                                                                        -
!-------------------------------------------------------------------------

System_Halt     no_op
                turn_off        SYS_OK
                f_decel         PEN
                f_decel         PSEUDO
Wait_Busy3      if_stat_on      PEN_BUSY,Wait_Busy3
                if_stat_on      PSEUDO_BUSY,Wait_Busy3

                drive_off       PEN

                sys_fault

!-------------------------------------------------------------------------
!                                                                        -
! Subroutine to configure the MAB map and the MCF map.                   -
!  Set the appropriate bits, and check the results.                      -
!                                                                        -
!-------------------------------------------------------------------------


Set_Config      let             map=0
                set_map         map
                set_bit         MABA_TALKER,map
                set_bit         MABA_LISTEN,map
                set_map         map
                get_map         map_back
                get_map_stat    map_stat

                if              map_stat<>0,Config_Fail
                if              map<>map_back,Config_Fail
```

```
Config_Pseudo  let            mcf=0
               set_mcf        PSEUDO,mcf

               set_bit        PSEUDO_MABA,mcf
               set_mcf        PSEUDO,mcf
               get_mcf        PSEUDO,mcf_back
               get_mcf_stat   PSEUDO,mcf_stat

               if             mcf_stat<>0,Config_Fail
               if             mcf<>mcf_back,Config_Fail

Exit_Config    return_sub

Config_Fail    sys_return

!-------------------------------------------------------------------------
!                                                                        -

! Subroutine to load cams into pen_cams array.                           -
!  Load each cam consecutively into the array.                           -
!  The last element in the array needs to indicate the end of the cam.   -
!  Execute a cam_data command to load the axis with the cam information.-
!  Immediately switch to a subset of the array which will be the         -
!   equivalent of the forward_cam table.                                 -
!-------------------------------------------------------------------------


Load_Cams      no_op
               let            ctr=0
               let            length=208
               let            last_el=length
Load_Forward   let_byte       temp=forward_cam[ctr]
               let_byte       pen_cams[ctr]=temp
               let            ctr=ctr+1
               if             ctr<last_el,Load_Forward

               let            ctr2=0
Load_Reverse   let_byte       temp=reverse_cam[ctr2]
               let_byte       pen_cams[ctr]=temp

               let            ctr=ctr+1
               let            ctr2=ctr2+1
               if             ctr2<last_el,Load_Reverse
               let_byte       pen_cams[ctr]=END_CAM

               cam_data       PEN,pen_cams,MS,DS
               let            start=0
               let            length=208
               switch_cam     PEN,start,length
               set_flag       IN_CAM_FWD

               return_sub
```

Clock Simulation

```
!-------------------------------------------------------------------------
! TITLE:       Clock Simulation
!
! DATE:        12/4/97
!
! DESCRIPTION: This program functions as a fictitious time clock
!              operating on either military or civilian time.  The
!              program is used to illustrate many of the arithmetic
!              functions of the controller.
!
! EQUIPMENT:   DeltaMax
!
! COMMANDS: clr_bit             declare             equ
!           gosub               goto                if
!           if_bit_clr          if_bit_set          if_tmr_on
!           integer             let                 no_op
!           return_sub          set_bit             set_tmr
!
! The above commands are commonly used in many programs.  This
! illustration provides the context and syntax required to successfuly
! integrate these commands into a complete program.  While the use of
! these commands varies from program to program, the following example
! will give the programmer an indication of a typical usage.
!-------------------------------------------------------------------------

              declare        on

!-------------------------------------------------------------------------
!                                                                        -
! Declare constants and variables                                        -
!                                                                        -
!-------------------------------------------------------------------------

TYPE          equ            0              ! Bit used to indicate
!                                               clock type
! TYPE=1 for Military; TYPE=0 for Civilian

TIMER         equ            72             ! Timer used for delays

clock         integer                       ! The variable used for
!                                               clock type
second        integer                       ! Counter indicates # of
!                                               seconds
minute        integer                       ! Counter indicates # of
!                                               minutes
hour          integer                       ! Counter indicates # of
!                                               hours
ticks         integer                       ! # of 10 msec clock ticks

!-------------------------------------------------------------------------
!                                                                        -
! Initialize variables                                                   -
!                                                                        -
!-------------------------------------------------------------------------
```

```
Defaults        let             second=0
                let             minute=0

! Using a 20 msec clock for demonstration purposes.

                let             ticks=2

!-------------------------------------------------------------------------
!                                                                        -
! Program Setup                                                          -
!  Check the type of clock previously used and toggle the bit.           -
!  The clock type alternates each time the program is restarted by       -
!   cycling power to the DeltaMax.                                       -
!  The program starts on a Military type clock whenever a reset command -
!   is issued from the Analyzer.                                         -
!                                                                        -
!-------------------------------------------------------------------------

Setup           no_op
Set_Clk_Type    if_bit_set      TYPE,clock,Set_Civilian

Set_Military    set_bit         TYPE,clock
                goto            Main

Set_Civilian    clr_bit         TYPE,clock
                let             hour=12

!-------------------------------------------------------------------------
!                                                                        -
! Main body of program.                                                  -
!  The clock counters are incremented appropriately.                     -
!  The hour is incremented based on the bit 'TYPE'.                      -
!                                                                        -
!-------------------------------------------------------------------------

Main            no_op

Start_Clk       no_op
Time_Seconds    gosub           One_Second
                let             second=second+1
if              second<60,Time_Seconds
                let             second=0

Time_Minutes    let             minute=minute+1
                if              minute<60,Time_Seconds
                let             minute=0

Time_Hours      no_op
                if_bit_clr      TYPE,clock,Civilian_Type
Military_Type   let             hour=hour+1
                if              hour<24,Time_Seconds
                let             hour=0
                goto            Time_Seconds

Civilian_Type   let             hour=hour+1
                if              hour<13,Time_Seconds
                let             hour=1
                goto            Time_Seconds
```

```
!-------------------------------------------------------------------------
!                                                                        -
! Subroutine used to delay a specified amount of time.                   -
!  For demonstration purposes timer will be 20 msec.                     -
!                                                                        -
!-------------------------------------------------------------------------

One_Second     set_tmr        TIMER,ticks
Wait_Sec       if_tmr_on      TIMER,Wait_Sec

               return_sub
```

DeviceNet without Parameter Object

```
!-------------------------------------------------------------------------
! TITLE:       DeviceNet without Parameter Object
!
! DESCRIPTION: This program runs a simple 'track_spd' to test a DeviceNet
!              connection. The CAN bus is configured for DeviceNet via the
!              'can_set' at address 1. It is intended that the user may stop
!              and start the motor over DeviceNet with DeviceNet input 1.
!              Once started the motor speed may be changed over DeviceNet by
!              writting to the spd variable in the program. The example EDS
!              file in the appendix is written to match this program.
!
! EQUIPMENT:   DeltaMax
!
! COMMANDS:    can_set              clr_dnet
!              dnet_range           if_dnet_on
!              if_dnet_off          set_dnet
!
! The above commands are commonly used in many programs.  This
! illustration provides the context and syntax required to successfuly
! integrate these commands into a complete program.  While the use of
! these commands varies from program to program, the following example
! will give the programmer an indication of a typical usage.
!-------------------------------------------------------------------------

              declare        on

DNET          equ            2

MOTOR         equ            1
BUSY          equ            78
DOWN          equ            77
OFF           equ            67
MOTOR_OFF     equ            (MOTOR*16)+OFF
MOTOR_BUSY    equ            (MOTOR*16)+BUSY
MOTOR_DOWN    equ            (MOTOR*16)+DOWN

DNET_ENABLE   equ            128           !equate for the DeviceNet enabled flag
DNET_FAULT    equ            129           !equate for the DeviceNet fault flag
DNET_INPUT1   equ            32            !equate for DeviceNet input 1 for do
                                            trackspd
DNET_OUTPUT1  equ            56            !equate for DeviceNet output 1 for
MOTOR_BUSY

!****** variables to have write access over DeviceNet
spd           integer
acdc          integer
multplr       integer
divisor       float

!**** delta_comp variables
lim           integer
pg            integer
ffwd          integer
```

```
!**** motor axis variables
com             integer
pos             integer

!**** can_set varables
canid           integer
canbaud         integer
canarb          integer

!**** application variables
angle           integer
div_angle       float
mult_angle      integer
can_faults      integer

!************DEVICENET INITIALIZATION*************************
                begin_cfg
                dnet_range      spd,divisor
                end_cfg
!************PROGRAM VARIABLE INITIALIZATIONS***************
    let             spd=100
                let             acdc=200

                let             divisor=1
                let             multplr=1
                let             canid=-1
                let             canbaud=-1
                let             canarb=DNET

                can_set         canid,canbaud,canarb
wait_dneten     if_stat_off     128,wait_dneten         ! verify devicenet enabled

!************MOTOR AXIS INITIALIZATION************************
                drive_on        MOTOR
                set_speed       MOTOR,spd
                set_ac_dc       MOTOR,acdc

!************SOFTWARE INTERRUPT INITIALIZATIONS***************
                swi_if_on       0,DNET_FAULT,CanFaulted
                enable_swi


!************MAIN PROGRAM Loop*********************************
Main            if_dnet_on      DNET_INPUT1,run_trackspd
                if_dnet_off     DNET_INPUT1,run_fdecel

run_trackspd    gosub           DoTrackSpeed
                goto            calcs
run_fdecel      gosub           ForceDecel

calcs           get_com         MOTOR,com
                get_pos         MOTOR,pos

                let             angle=pos&4095              ! get present angle from
pos
                let             div_angle=angle/divisor     ! divide angle
```

```
               let             mult_angle=angle*multplr    ! multiple angle

               if_stat_on      MOTOR_BUSY,set_out1
               clr_dnet        DNET_OUTPUT1
               goto            Main
set_out1       set_dnet        DNET_OUTPUT1
               goto            Main


!************************************************************
!******************SUBROUTINES*******************************
!************************************************************

!**** Force Decel Subr.*******
ForceDecel     if_stat_off     MOTOR_BUSY,skipfdecel
               f_decel         MOTOR
skipfdecel     return_sub

!**** Track Speed Subr.*******
DoTrackSpeed
               if_stat_off     MOTOR_OFF,gotrack
               drive_on        MOTOR

gotrack        set_ac_dc       1,acdc
               track_spd       MOTOR,spd

               return_sub


!************************************************************
!******************INTERRUPT ROUTINES************************
!************************************************************

!**** CanBus faulted
!****  Description: Tries to clear bus_off condition due to a
!                   possible DeviceNet Network error, for a
!                   limited number of times.
CanFaulted     disable_swi
               let             can_faults=can_faults+1
               if              can_faults>10,skipcanset
               can_set         canid,canbaud,canarb
skipcanset     enable_swi
               return_sub
```

DeviceNet with Parameter Object

```
!-------------------------------------------------------------------------
! TITLE:        DeviceNet with Parameter Object
!
! DESCRIPTION:  This program runs a simple 'track_spd' to test a DeviceNet
!               connection. A DeviceNet Parameter Object is programmed at the
!               beginning of the program. It is intended that a DeviceNet
!               Manager Utility program may automatically creat an EDS file
!               from this example program running in a DeltaMax. A Parameter
!               Object in a DeviceNet device is readable by such Utility
!               programs as Allen Bradley's DeviceNet Manager™.
!
! EQUIPMENT:    DeltaMax
!
! COMMANDS:     begin_cfg             dnet_flag_pm
!               dnet_flt_pm           dnet_int_pm
!               dnet_p_code           end_cfg
!
! The above commands are commonly used in many programs.  This
! illustration provides the context and syntax required to successfuly
! integrate these commands into a complete program.  While the use of
! these commands varies from program to program, the following example
! will give the programmer an indication of a typical usage.
!-------------------------------------------------------------------------

               declare        on

DNET           equ            2

MOTOR          equ            1
BUSY           equ            78
DOWN           equ            77
OFF            equ            67
DRV1_OFF       equ            83
MOTOR_OFF      equ            (MOTOR*16)+OFF
MOTOR_BUSY     equ            (MOTOR*16)+BUSY
MOTOR_DOWN     equ            (MOTOR*16)+DOWN

DNET_ENABLE    equ            128           !equate for the DeviceNet enabled flag
DNET_FAULT     equ            129           !equate for the DeviceNet fault flag
DNET_INPUT1    equ            32            !equate for DeviceNet input 1 for do
                                              trackspd
DNET_INPUT2    equ            33            !equate for DNet Input 2 to turn drive
off
DNET_OUTPUT1   equ            56            !equate for DeviceNet output 1 for
MOTOR_BUSY

RUN_BIT        equ            255

! variables to have write access over DeviceNet
spd            integer
acdc           integer
time           integer
multplr        float
divisor        float
```

```
! motor axis variables
com             integer
pos             integer
time_strt       integer
get_tm          integer
tmp_tm          integer

run_text        text            "run input"
div_text        text            "float divisor"
spd_text        text            "speed in rpm"
acdc_text       text            "accel"
time_text       text            "runtime (msec)"
ID_text         text            "Dnet Eamle"
DrvOff_text     text            "MonetaryDriveOff"


!************DEVICENET PARAMETER INITIALIZATION**************

                begin_cfg                           !this is the first executable
                                                     instr.
                dnet_p_code     10,ID_text          !define devicenet product code
                dnet_range      spd,divisor         !define read/write memory for
devicenet
                dnet_int_pm     time,time_text,10,100,10
                dnet_flt_pm     divisor,div_text,1.0,100.0,10.5
                dnet_int_pm     spd,spd_text,10,1000,100
                dnet_int_pm     acdc,acdc_text,10,100,50
                dnet_flag_pm    DNET_INPUT1,run_text
                dnet_flag_pm    DNET_INPUT2,DrvOff_text
                can_set         -1,125,2
                end_cfg

!************DEFINE SWI(s)**********************************
                swi_if_on       0,DNET_INPUT1,TurnDriveOn
                swi_if_off      2,DNET_INPUT1,Stop
                swi_if_on       1,DNET_INPUT2,TurnDriveOff

                enable_swi
!************PROGRAM VARIABLE INITIALIZATIONS***************
                let             multplr=1
                get_time        time_strt


!************PROGRAM START**********************************
                drive_on        1

wait_dneten     if_stat_off     128,wait_dneten        ! verify devicenet enabled


loop            get_time        get_tm
                let             tmp_tm = get_tm - time_strt
                let             tmp_tm = tmp_tm * 5
                if_flag_on      RUN_BIT,Trackspeed
                let             time = 0
                goto            loop
```

```
Trackspeed     let            time = tmp_tm
               set_ac_dc      1,acdc
               track_spd      1,spd
               goto           loop

done           goto           done

!**********SWI: TurnDriveOn***********************************
TurnDriveOn    if_stat_off    DRV1_OFF,skip_drvon
               drive_on       1
skip_drvon     get_time       time_strt
               set_flag       RUN_BIT
               enable_swi
               return_sub

Stop           f_decel        1
               clr_flag       RUN_BIT
               enable_swi
               return_sub

TurnDriveOff   drive_off      1
               clr_flag       RUN_BIT
               enable_swi
               return_sub
```

Home Sequence

```
!-------------------------------------------------------------------------
! TITLE:       Home Sequence
!
! DATE:        12/5/97
!
! DESCRIPTION: This program illustrates a homing sequence for a resolver
!              based system.  After the homing sequence has been
!              initiated, the operator can execute either an index
!              or a positional move.
!
! EQUIPMENT:   DeltaMax
!
! COMMANDS: disable_swi         index           jog_ccw
!           jog_cw              mod_index       mod_position
!           position            set_ac_dc       set_gl_ccw
!           set_gl_cw           set_speed       set_swi_mask
!
! The above commands are commonly used in many programs.  This
! illustration provides the context and syntax required to successfuly
! integrate these commands into a complete program.  While the use of
! these commands varies from program to program, the following example
! will give the programmer an indication of a typical usage.
!-------------------------------------------------------------------------

              declare        on

!-------------------------------------------------------------------------
!                                                                       -
! Declare constants and variables                                       -
!                                                                       -
!-------------------------------------------------------------------------


!
!  Axis related constants
!

XFR           equ            1              ! Transfer axis
BUSY          equ            78             ! busy flag offset
DOWN          equ            77             ! down flag offset
XFR_BUSY      equ            (XFR*16)+BUSY  ! busy flag - Transfer
XFR_DOWN      equ            (XFR*16)+DOWN  ! down flag - Transfer


!
!  I/O and Flags
!

INIT          equ            0              ! Initialize axis (input)
INDEX         equ            1              ! Start index (input)

POSITION      equ            2              ! Position to zero (input)
MOD_IDX       equ            3              ! Start modified index (input)
MOD_POS       equ            4              ! Start modified position (input)
```

```
DIR             equ             5               ! ON=CCW; OFF=CW (input)
ZREV            equ             6               ! Zero rev switch (input)
RESET           equ             7               ! Reset Fault (input)
SYS_OK          equ             16              ! System okay (output)

TIMER           equ             72              ! Timer used for delays

INIT_CMPT       equ             255             ! Flag for initialization
!                                                 complete

!
! Motion related constants
!

INIT_SPD        equ             200             ! Initialize jog speed
INIT_ACDC       equ             200             ! Initialize accel/decel
!                                                 rate
IDX_SPD         equ             800             ! Index speed
IDX_ACDC        equ             300             ! Index accel/decel rate
POS_SPD         equ             300             ! Position speed
POS_ACDC        equ             200             ! Position accel/decel rate
IDX_DIST        equ             40960           ! Index distance = 10 turns
IDX_DIST1       equ             20480           ! Index distance = 5 turns
HOME            equ             0               ! Absolute home
POS1            equ             -20480          ! position to -5 turns
DEG             equ             4               ! degree for modified profiles


!
! Variable definitions
!

time            integer         ! Determines the delay time for a timer.
swi_mask        integer         ! Determines the active software
!                                 interrupts.

!-------------------------------------------------------------------------
!                                                                        -
! Initialize variables                                                   -
!                                                                        -
!-------------------------------------------------------------------------

Defaults        let             time=100
                let             swi_mask=63

!-------------------------------------------------------------------------
!                                                                        -
! Program Setup                                                          -
!  Define software interrupts.                                           -
!                                                                        -
!-------------------------------------------------------------------------

Setup           swi_if_on       0,INIT,Home_Xfr
                swi_if_on       1,INDEX,Index_Xfr
                swi_if_on       2,POSITION,Position_Xfr
                swi_if_on       3,MOD_IDX,Mod_Idx_Xfr
                swi_if_on       4,MOD_POS,Mod_Pos_Xfr
                swi_if_on       5,XFR_DOWN,Fault

                enable_swi
```

```
!-------------------------------------------------------------------------
!                                                                        -
! Main body of program.                                                  -
!  Enable XFR.                                                           -
!  Remain in Loop, waiting for software interrupts.                      -
!                                                                        -
!-------------------------------------------------------------------------

Main           clr_flag       INIT_CMPT
               turn_on        SYS_OK
               drive_on       XFR
               delta_comp     XFR,1.0,1.0,30,0,1.0,6000

Loop           goto           Loop

!-------------------------------------------------------------------------
!                                                                        -
! Software Interrupt Routine to Home XFR axis.                           -
!  Setup software interrupt mask to only allow XFR_DOWN to be active.    -
!  Define XFR motion parameters.                                         -
!  Test to determine direction of home sequence.                         -
!  Test ZREV switch for current state.                                   -
!    If on the switch, must jog off switch to start.                     -
!  Jog toward the ZREV switch and monitor the state of the IO.           -
!  When ZREV turns on, stop and set a global zero according to the       -
!   HOME direction.                                                      -
!  Positon the XFR to HOME.                                              -
!  Set the flag INIT_CMPT.                                               -
!  Enable software interrupts and return.                                -
!                                                                        -
!-------------------------------------------------------------------------

Home_Xfr       let            swi_mask=32
               set_swi_mask   swi_mask
               enable_swi


               set_speed      XFR,INIT_SPD
               set_ac_dc      XFR,INIT_ACDC
               if_io_on       DIR,Find_Home_Ccw

Find_Home_Cw   if_io_off      ZREV,Off_Switch_Cw
               jog_ccw        XFR

On_Switch_Cw   if_stat_on     XFR_DOWN,Fault
               if_io_on       ZREV,On_Switch_Cw
               f_decel        XFR
               gosub          Wait_Stop

Off_Switch_Cw  jog_cw         XFR
               if_stat_on     XFR_DOWN,Fault
               if_io_off      ZREV,Off_Switch_Cw
               set_gl_cw      XFR
               f_decel        XFR
               gosub          Wait_Stop
               goto           Xfr_At_Home
```

```
Find_Home_Ccw  if_io_off      ZREV,Off_Switch_Ccw
               jog_cw         XFR

On_Switch_Ccw  if_stat_on     XFR_DOWN,Fault
               if_io_on       ZREV,On_Switch_Ccw
               f_decel        XFR
               gosub          Wait_Stop

Off_Switch_Ccw jog_ccw        XFR

               if_stat_on     XFR_DOWN,Fault
               if_io_off      ZREV,Off_Switch_Ccw
               set_gl_ccw     XFR
               f_decel        XFR
               gosub          Wait_Stop

Xfr_At_Home    position       XFR,HOME
               gosub          Wait_Stop

Exit_Home_Xfr  set_flag       INIT_CMPT
               disable_swi
               let            swi_mask=63
               set_swi_mask   swi_mask
               enable_swi
               return_sub


!-------------------------------------------------------------------------
!                                                                        -
! Software Interrupt Routine to Index the XFR 10 turns.                  -
!  Setup software interrupt mask to only allow XFR_DOWN to be active.    -
!  Check to see that INIT_CMPT has been set.                             -
!    If not skip index.                                                  -
!  Define axis motion parameters.                                        -
!  Index XFR 10 turns.                                                   -
!  Wait for index to complete.                                           -
!  Enable software interrupts                                            -
!                                                                        -
!-------------------------------------------------------------------------

Index_Xfr      let            swi_mask=32
               set_swi_mask   swi_mask
               enable_swi


               if_flag_off    INIT_CMPT,Exit_Index
               set_speed      XFR,IDX_SPD
               set_ac_dc      XFR,IDX_ACDC
               index          XFR,IDX_DIST
               gosub          Wait_Stop
Exit_Index     disable_swi
               let            swi_mask=63
               set_swi_mask   swi_mask
               enable_swi
               return_sub
```

```
!-------------------------------------------------------------------------
!                                                                        -
! Software Interrupt Routine to Position XFR to HOME.                    -
!  Setup software interrupt mask to only allow XFR_DOWN to be active.    -
!  Check to see that INIT_CMPT has been set.                             -
!    If not skip position.                                               -
!  Define axis motion parameters.                                        -
!  Position XFR to HOME.                                                 -
!  Wait for index to complete.                                           -
!  Enable software interrupts                                            -
!                                                                        -
!-------------------------------------------------------------------------

Position_Xfr   let            swi_mask=32
               set_swi_mask   swi_mask
               enable_swi

               if_flag_off    INIT_CMPT,Exit_Position
               set_speed      XFR,POS_SPD
               set_ac_dc      XFR,POS_ACDC

               position       XFR,HOME
               gosub          Wait_Stop

Exit_Position  disable_swi
               let            swi_mask=63
               set_swi_mask   swi_mask
               enable_swi
               return_sub

!-------------------------------------------------------------------------
!                                                                        -
! Software Interrupt Routine to use a modified index to XFR 5 turns.     -
!  Setup software interrupt mask to only allow XFR_DOWN to be active.    -
!  Check to see that INIT_CMPT has been set.                             -
!    If not skip index.                                                  -
!  Define axis motion parameters.                                        -
!  Index XFR 5 turns.                                                    -
!  Wait for index to complete.                                           -
!  Enable software interrupts                                            -
!                                                                        -
!-------------------------------------------------------------------------

Mod_Idx_Xfr    let            swi_mask=32
               set_swi_mask   swi_mask
               enable_swi


               if_flag_off    INIT_CMPT,Exit_Mod_Idx
               mod_index      XFR,IDX_SPD,IDX_ACDC,IDX_DIST1,DEG
               gosub          Wait_Stop

Exit_Mod_Idx   disable_swi
               let            swi_mask=63
               set_swi_mask   swi_mask
               enable_swi
               return_sub
```

```
!-------------------------------------------------------------------------
!                                                                        -
! Software Interrupt Routine to Position XFR to -5 turns using a        -
!    modified profile.                                                   -
!  Setup software interrupt mask to only allow XFR_DOWN to be active.    -
!  Check to see that INIT_CMPT has been set.                             -
!    If not skip position.                                               -
!  Define axis motion parameters.                                        -
!  Position XFR to -5 turns.                                             -
!  Wait for index to complete.                                           -
!  Enable software interrupts                                            -
!                                                                        -
!-------------------------------------------------------------------------

Mod_Pos_Xfr    let            swi_mask=32
               set_swi_mask   swi_mask
               enable_swi

               if_flag_off    INIT_CMPT,Exit_Mod_Pos
               set_speed      XFR,POS_SPD
               set_ac_dc      XFR,POS_ACDC

               mod_position   XFR,POS_SPD,POS_ACDC,POS1,DEG
               gosub          Wait_Stop

Exit_Mod_Pos   disable_swi
               let            swi_mask=63
               set_swi_mask   swi_mask
               enable_swi
               return_sub

!-------------------------------------------------------------------------
!                                                                        -
! Subroutine to wait for motion to be completed.                        -
!                                                                        -
!-------------------------------------------------------------------------

Wait_Stop      if_stat_on     XFR_BUSY,Wait_Stop  ! check for axis busy
               return_sub

!-------------------------------------------------------------------------
!                                                                        -
! Software Interrupt Routine used for a XFR fault condition.            -
!  Turn off the System Ok output.                                        -
!  Check to see that all motion has stopped.                             -
!  Wait for reset input, then restart at Main.                           -
!                                                                        -
!-------------------------------------------------------------------------
```

```
Fault           no_op
                turn_off        SYS_OK
                f_decel         XFR

                set_tmr         TIMER,200
Wait_Fault      if_tmr_on       TIMER,Wait_Fault


Start_Over      if_io_off       RESET,Start_Over
                drive_off       XFR
                let             swi_mask=63
                set_swi_mask    swi_mask
                enable_swi
                restart_at      Main
```

Index with over_draw

```
!-------------------------------------------------------------------------
! TITLE:       Index with over_draw
!
! DATE:        12/8/97
!
! DESCRIPTION: This program illustrates the two overdraw modes and the
!              differences between them.  The operator can observe
!              these differences by manually selecting the overdraw
!              mode and then triggering the sensor input during the
!              execution of the index.
!
! EQUIPMENT:   DeltaMax
!
! COMMANDS: clr_local           disable_hwi         drive_off
!           drive_on            enable_hwi          get_com
!           get_pos             get_trap_pos        if_io_on
!           if_stat_off         over_draw           set_local
!           set_ovd_mode
!
! The above commands are commonly used in many programs.  This
! illustration provides the context and syntax required to successfuly
! integrate these commands into a complete program.  While the use of
! these commands varies from program to program, the following example
! will give the programmer an indication of a typical usage.
!-------------------------------------------------------------------------

              declare        on

!-------------------------------------------------------------------------
!                                                                        -
! Declare constants and variables                                        -
!                                                                        -
!-------------------------------------------------------------------------

!
!  Axis related constants
!

FEED          equ            1

BUSY          equ            78                 ! busy flag offset
DOWN          equ            77                 ! down flag offset
ARMED         equ            75                 ! HWI armed flag
!                                                  offset
INDEXING      equ            73                 ! indexing flag offset

FEED_BUSY     equ            (FEED*16)+BUSY     ! busy flag - FEED
FEED_DOWN     equ            (FEED*16)+DOWN     ! down flag - FEED
FEED_ARMED    equ            (FEED*16)+ARMED    ! HWI armed flag -
!                                                  FEED
FEED_INDEXING equ            (FEED*16)+INDEXING ! indexing flag -
!                                                  FEED
```

```
OD_MODE_0       equ             0       ! Overdraw mode - sensor always
!                                         active

OD_MODE_1       equ             1       ! Overdraw mode - sensor only
!                                         active during porch
!
!  I/O and Flags
!

SENSOR          equ             0               ! Photo sensor (input)
INDEX           equ             1               ! Start index (input)
OVD_MODE        equ             2               ! Select overdraw mode
!                                                 (input)
RESET           equ             6               ! Reset Fault (input)

SYS_OK          equ             16              ! System okay (output)

TIMER           equ             72              ! Timer used for delays.

!
!  Variable definitions
!

com             integer         ! Stores the current 'get_com' value.
pos             integer         ! Stores the current 'get_pos' value.
tpos            integer         ! Stores the current 'get_trap_pos' value.
diff            integer         ! Variable used to measure differences.
end_com         integer         ! Used to store the ending com value.

end_pos         integer         ! Used to store the ending pos value.
temp            integer         ! Used as a temporary storage location.
idx_dist        integer         ! The index distance.
ovd_speed       integer         ! The speed at which the search occurs.
srch_dist       integer         ! The distance to search for sensor.
post_dist       integer         ! The distance travelled after the sensor
!                                 trips.

!-------------------------------------------------------------------------
!                                                                        -
! Initialize variables                                                   -
!                                                                        -
!-------------------------------------------------------------------------

Defaults        let             ovd_speed=25
                let             srch_dist=8192
                let             post_dist=1024
                let             idx_dist=40960


!-------------------------------------------------------------------------
!                                                                        -
! Program Setup                                                          -
!  Define FEED motion parameters.                                        -
!                                                                        -
!-------------------------------------------------------------------------
```

```
Setup           set_ac_dc       FEED,50

                set_speed       FEED,200

!-------------------------------------------------------------------------
!                                                                        -
! Main body of program                                                   -
!  Enable DRAW.                                                          -
!  Remain in Loop, waiting for transition on INDEX.                      -
!                                                                        -
!-------------------------------------------------------------------------

Main            drive_on        FEED

Loop            if_io_on        INDEX,Loop
Loop1           if_io_off       INDEX,Loop1
                if_io_on        OVD_MODE,Mode_1

!-------------------------------------------------------------------------
!                                                                        -
! Routine for overdraw mode 0 (HWI is always active).                    -
!  Clear any local zero.                                                 -
!  Check status of HWI armed flag.                                       -
!    If flag on disarm HWI, otherwise continue.                          -
!  Setup the HWI for an overdraw of mode 0.                              -
!  Set a local zero in order to have an accurate start point.            -
!  Make sure the FEED HWI is armed.                                      -
!  Index the FEED and wait until motion is complete.                     -
!  Based on the status of the FEED_ARMED flag, determine if the          -
!   distances are correct.                                               -
!                                                                        -
!-------------------------------------------------------------------------

Mode_0          clr_local       FEED
                if_stat_off     FEED_ARMED,No_Disable
                disable_hwi     FEED

No_Disable      set_ovd_mode    FEED,OD_MODE_0
                set_local       FEED

                enable_hwi
                over_draw       FEED,ovd_speed,srch_dist,post_dist


Not_Armed       if_stat_off     FEED_ARMED,Not_Armed

                index           FEED,idx_dist
                gosub           Wait_Stop

                if_stat_on      FEED_ARMED,Check_Dist
                get_com         FEED,end_com
                if              end_com>=idx_dist,Loop
                turn_off        SYS_OK
                sys_fault
```

```
Check_Dist      get_com         FEED,end_com
                let             temp=idx_dist+srch_dist
                if              end_com=temp,Loop

                turn_off        SYS_OK
                sys_fault

!-------------------------------------------------------------------------
!                                                                        -
! Routine for overdraw mode 1 (HWI is only active in overdraw porch).   -
!  Clear any local zero.                                                -
!  Check status of HWI armed flag.                                      -
!    If flag on disarm HWI, otherwise continue.                         -
!  Setup the HWI for an overdraw of mode 1.                             -
!  Set a local zero in order to have an accurate start point.           -
!  Make sure the FEED HWI is armed.                                     -
!  Index the FEED and wait until motion is complete.                    -
!  Based on the status of the FEED_ARMED flag, determine if the         -
!   distances are correct.                                              -
!                                                                       -
!-------------------------------------------------------------------------

Mode_1          clr_local       FEED
                if_stat_off     FEED_ARMED,No_Disable1
                disable_hwi     FEED

No_Disable1     set_ovd_mode    FEED,OD_MODE_1
                set_local       FEED
                get_pos         FEED,pos

                enable_hwi
                over_draw       FEED,25,8192,1024

Not_Armed1      if_stat_off     FEED_ARMED,Not_Armed1

                index           FEED,40960

                gosub           Wait_Stop

                if_stat_on      FEED_ARMED,Check_Dist1
                get_com         FEED,end_com
                get_trap_pos    FEED,tpos
                let             diff=end_com-tpos
                if              diff=1024,Loop
                turn_off        SYS_OK
                sys_return

Check_Dist1     get_com         FEED,end_pos
                let             temp=idx_dist+srch_dist

                if              end_pos<>temp,Check_Dist1
                if              end_pos=temp,Loop
                turn_off        SYS_OK
                sys_return
```

```
!-------------------------------------------------------------------------
!                                                                        -
! Routine used for a FEED fault condition.                               -
!  Turn off the System Ok output.                                        -
!  Check to see that all motion has stopped.                             -
!  Wait for reset input, then restart at Main.                           -
!                                                                        -
!-------------------------------------------------------------------------

Fault          no_op
               turn_off       SYS_OK
               f_decel        FEED

               set_tmr        TIMER,200
Wait_Fault     if_tmr_on      TIMER,Wait_Fault

Start_Over     if_io_off      RESET,Start_Over
               drive_off      FEED

               restart_at     Main

!-------------------------------------------------------------------------
!                                                                        -
! Routine used to wait for motion to stop on FEED.                       -
!  Check the DOWN and INDEXING flags, and when index is complete return.-
!                                                                        -
!-------------------------------------------------------------------------


Wait_Stop      no_op
               if_stat_on     FEED_DOWN,Fault
               if_stat_on     FEED_INDEXING,Wait_Stop
               return_sub
```

Programmable Limit Switch Functions

```
!-------------------------------------------------------------------------
! TITLE:       Programmable Limit Switch Functions
!
! DATE:        12/8/97
!
! DESCRIPTION: This program demonstrates the Programmable Limit Switch
!              (PLS) functions.  These functions include defining PLS,
!              PLS masking and creating an angular offset. Each of these
!              functions can be visually observed during program
!              execution.
!
! EQUIPMENT:   DeltaMax
!
! COMMANDS: clr pls           get_pls_ang          get_pls_mask
!           get_pls_out       l track spd          preset
!           set_pls_ang       set_pls_cnt          set_pls_mask
!           track_spd
!
!
! The above commands are commonly used in many programs.  This
! illustration provides the context and syntax required to successfuly
! integrate these commands into a complete program.  While the use of
! these commands varies from program to program, the following example
! will give the programmer an indication of a typical usage.
!-------------------------------------------------------------------------

              declare        on

!-------------------------------------------------------------------------
!                                                                        -
! Declare constants and variables                                        -
!                                                                        -
!-------------------------------------------------------------------------

!
! Axis related constants
!

PSEUDO         equ            2
PLS            equ            2

CALC_PLS       equ            106            ! Calculating flag on PLS

PSEUDO_MABA    equ            10             ! PSEUDO sends data to
!                                               MABA
PLS_MABA       equ            16             ! PLS receives data from
!                                               MABA

MASTER         equ            30             ! Define PSEUDO as master
SLAVE          equ            22             ! Define PLS as slave

!
!  I/O and Flags
!
```

```
DO_MASK         equ             0               ! Define output mask (input)

CLR_PLS         equ             1               ! Create an offset (input)
OFFSET          equ             2               ! Create an offset (input)
SYS_OK          equ             23              ! System okay (output)

TIMER           equ             72              ! Timer used for delays

FIRST_TIME      equ             251             ! Flag indicating first
!                                                 time

!
!  Variable definitions
!

map             integer         ! Used to configure the MASTER/SLAVE on
!                                 the MAB.
map_back        integer         ! Return variable for the current 'map'
!                                 setup.
map_stat        integer         ! Return variable for the current 'map'
!                                 status.
mcf             integer         ! Used to configure the PSEUDO.
mcf_stat        integer         ! Current multi-function control
!                                 status
mcf_back        integer         ! Return variable for the current 'mcf'
!                                 setup.
on_angle        integer         ! The angle to turn on a module.

off_angle       integer         ! The angle to turn off a module.
hw_pls          integer         ! Used to define modules 16-19.
sw_pls          integer         ! Used to define modules 24-27.
clr_this_pls    integer         ! Used to clear modules 16-19 and 24-27.
out_stat        integer         ! Return variable for 'get_pls_out'.
angle           integer         ! Return variable for 'get_angle'.
pls_cnt         integer         ! Define the counts per PLS cycle
time            integer         ! Determines the delay time for a timer.
mask            integer         ! Used to mask off specific pls modules.
com             integer

mask_back       integer         ! Return variable for 'get_pls_mask'.
offset          integer         ! The amount of offset added to the MAB.

!-------------------------------------------------------------------------
!                                                                        -
! Initialize variables                                                   -
!                                                                        -
!-------------------------------------------------------------------------

Defaults        let             on_angle=0
                let             off_angle=1023
                let             hw_pls=16
                let             sw_pls=24
                let             clr_this_pls=16
                let             pls_cnt=8192
                let             mask=4294901760
```

```
!-------------------------------------------------------------------------
!                                                                        -
! Program Setup                                                          -
!  Define the PLS parameters.                                            -
!  Define axis motion parameters.                                        -
!  Define software interrupts.                                           -
!                                                                        -
!-------------------------------------------------------------------------

               gosub          Set_Config
               set_pls_cnt    PLS,pls_cnt
               gosub          Wait_Calc
Define_Pls

               set_pls_ang    PLS,on_angle,off_angle,hw_pls

               gosub          Wait_Calc

               set_pls_ang    PLS,on_angle,off_angle,sw_pls
               gosub          Wait_Calc

               let            on_angle=on_angle+1024
               let            off_angle=off_angle+1024
               let            hw_pls=hw_pls+1
               let            sw_pls=sw_pls+1
               if             hw_pls<20,Define_Pls

               set_ac_dc      PSEUDO,20

               swi_if_on      0,DO_MASK,Define_Mask
               swi_if_on      1,CLR_PLS,Clear_Pls
               swi_if_on      2,OFFSET,Create_Offset
               enable_swi

               set_pls_mask   PLS,mask


!-------------------------------------------------------------------------
!                                                                        -
! Main body of program                                                   -
!  Start motion.                                                         -
!  Get current angle and PLS output information.                         -
!  Remain in Loop, waiting for software interrupts.                     -
!                                                                        -
!-------------------------------------------------------------------------

Main           no_op
               turn_on        SYS_OK
               track_spd      PSEUDO,100
!              l_track_spd    PSEUDO,100
!
! This command could be used in place of a 'track_spd' when slow
! or fractional speeds are required.

Loop           get_pls_out    PLS,out_stat
               get_com        PSEUDO,com
               get_pls_ang    PLS,angle
               gosub          Delay
               goto           Loop
```

```
!-------------------------------------------------------------------------
!                                                                        -
! Software Interrupt Routine used to define a PLS output mask.          -
!  Test to see if this is the first time executing this routine.         -
!  Setup the PLS mask.                                                   -
!  Check for correct mask.                                               -
!    If mask is incorrect then stop system.                              -
!  Enable the software interrupts and return.                            -
!                                                                        -
!-------------------------------------------------------------------------

Define_Mask    no_op
               if_flag_on     FIRST_TIME,Skip_First
               let            mask=0
               set_bit        16,mask
               set_flag       FIRST_TIME

Skip_First     set_pls_mask   PLS,mask
               get_pls_mask   PLS,mask_back
               if             mask<>mask_back,Fault_Mask

               let            mask=mask{{1
               if_bit_set     20,mask,Start_Hw_Mask
               if_bit_set     28,mask,Start_Sw_Mask
               goto           Exit_Mask

Start_Sw_Mask  clr_bit        28,mask
               set_bit        16,mask
               goto           Exit_Mask

Start_Hw_Mask  clr_bit        20,mask
               set_bit        24,mask
               goto           Exit_Mask


Fault_Mask     turn_off       SYS_OK
               sys_return

Exit_Mask      enable_swi
               return_sub

!-------------------------------------------------------------------------
!                                                                        -
! Software Interrupt Routine to create an offset on the MAB.            -
!  Define the amount of offset.                                          -
!  Enable software interrupts and return.                                -
!                                                                        -
!-------------------------------------------------------------------------

Create_Offset  no_op
               preset         PLS,offset
               gosub          Wait_Calc

               enable_swi
               return_sub
```

```
!-------------------------------------------------------------------------
!                                                                        -
! Software Interrupt Routine to clear the next PLS being used.          -
!  First clear 16-19, then 24-27.                                       -
!  Enable software interrupts and return.                               -
!                                                                        -
!-------------------------------------------------------------------------

Clear_Pls      no_op
               if             clr_this_pls<28,Do_Clear_Pls
               let            clr_this_pls=16

Do_Clear_Pls   select         clr_this_pls
                 case           16
                   clr_pls      2,16
                   exit_select
                 case           17
                   clr_pls      2,17
                   exit_select
                 case           18
                   clr_pls      2,18
                   exit_select
                 case           19
                   clr_pls      2,19
                   exit_select

                 case           24
                   clr_pls      2,24
                   exit_select
                 case           25
                   clr_pls      2,25
                   exit_select
                 case           26
                   clr_pls      2,26
                   exit_select
                 case           27
                   clr_pls      2,27
                   exit_select
               end_select



               let            clr_this_pls=clr_this_pls+1

               if             clr_this_pls<20,Exit_Clear_Pls
               let            clr_this_pls=24

Exit_Clear_Pls enable_swi
               return_sub
```

```
!-------------------------------------------------------------------------
!                                                                        -
! Subroutine to configure the MAB map and the MCF map.                   -
!  Set the appropriate bits, and check the results.                      -
!                                                                        -
!-------------------------------------------------------------------------

Set_Config     no_op
               let             mcf=0
               set_mcf         PSEUDO,mcf
               set_bit         PSEUDO_MABA,mcf
               set_bit         PLS_MABA,mcf
               set_mcf         PSEUDO,mcf
               get_mcf         PSEUDO,mcf_back
               get_mcf_stat    PSEUDO,mcf_stat

               if              mcf<>mcf_back,Config_Fail
               if              mcf_stat<>0,Config_Fail

               let             map=0
               set_map         map
               set_bit         MASTER,map
               set_bit         SLAVE,map
               set_map         map

Exit_Config    return_sub

Config_Fail    turn_off        SYS_OK
               sys_return

!-------------------------------------------------------------------------
!                                                                        -
! Subroutine to monitor the PLS CALCULATING flag.                        -
!  Wait until the card is finished calculating.                          -
!                                                                        -
!-------------------------------------------------------------------------

Wait_Calc      no_op
               if_stat_on      CALC_PLS,Wait_Calc
               return_sub

!-------------------------------------------------------------------------
!                                                                        -
! Subroutine used to create a time delay.                                -
!  Wait until the timer times out.                                       -
!                                                                        -
!-------------------------------------------------------------------------

Delay          set_tmr         TIMER,time
Wait_Time      if_tmr_on       TIMER,Wait_Time
               return_sub
```

Piecewise Profile

```
!-------------------------------------------------------------------------
! TITLE:       Piecewise Profile
!
! DATE:        12/8/97
!
! DESCRIPTION: This program illustrates the use of a piecewise profile,
!              and the methods by which to execute a profile.  The
!              time to execute the selected profile is compared to the
!              calculated time as a measure of performance.
!
! EQUIPMENT:   DeltaMax
!
! COMMANDS: exec_profile         get_ms_time         get_pstat
!           get_time             prep_profile        set_trig_pw
!
! The above commands are commonly used in many programs.  This
! illustration provides the context and syntax required to successfuly
! integrate these commands into a complete program.  While the use of
! these commands varies from program to program, the following example
! will give the programmer an indication of a typical usage.
!-------------------------------------------------------------------------

              declare        on

!-------------------------------------------------------------------------
!                                                                        -
! Declare constants and variables                                        -
!                                                                        -
!-------------------------------------------------------------------------

!
!  Axis related constants
!

FEED          equ            1
MASTER        equ            2

BUSY          equ            78              ! busy flag offset
DOWN          equ            77              ! down flag offset
CALC          equ            74              ! calculating flag offset
LOCK          equ            79              ! lock flag offset
FEED_BUSY     equ            (FEED*16)+BUSY  ! busy flag - FEED

FEED_DOWN     equ            (FEED*16)+DOWN  ! down flag - FEED
FEED_CALC     equ            (FEED*16)+CALC  ! calculating flag - FEED
FEED_LOCK     equ            (FEED*16)+LOCK  ! lock flag - FEED

LK_MODE_3     equ            3              ! lock mode 3
MABA_TALKER   equ            30             ! MASTER is talker on MABA
MABA_LISTEN   equ            23             ! FEED is listener on MABA
PSEUDO_MABA   equ            10             ! PSEUDO sends data to MABA
```

```
!
!  I/O and Flags
!

EXEC_PROF       equ             0               ! Start profile execution
!                                                 (input)
EXEC_TYPE       equ             1               ! Execution type (input)
!                                                 ON=Lock;OFF=Normal
RESET           equ             6               ! Reset Fault (input)
SYS_OK          equ             16              ! System okay (output)

TIMER           equ             72              ! Timer used for delays.


!
!  Variable Definitions
!

start_angle     integer         ! The start angle for the Lock to begin.
temp            integer         ! Used as a temporary storage variable.
prof_stat       integer         ! Stores the result of a get_pstat.
diff            integer         ! Stores the difference between two
!                                 variables.
error           integer         ! The error between measured and
!                                 calculated time.
exec_time       integer         ! The calculated execution time.

time            integer         ! Determines the delay time for a timer.
strt_time       integer         ! Stores the time value at start of
!                                 profile.
strt_ms_tm      integer
end_ms_tm       integer
diff_ms         integer

end_time        integer         ! Stores the time value at end of profile.
map             integer         ! Used to configure the MASTER/SLAVE on
!                                 the MAB.
map_back        integer         ! Returns the current 'map' setup.
map_stat        integer         ! Return the current 'map' status.

mcf             integer         ! Used to configure the PSEUDO.
mcf_back        integer         ! Returns the current 'mcf setup.
mcf_stat        integer         ! Return the current 'mcf' status.

!
!  Data array for piecewise profile
!

prodata         begin_data
                data            500,250,8192
                data            1000,250,8192
                data            250,100,8192
                data            0,100,8192
                end_data
```

```
!-------------------------------------------------------------------------
!                                                                        -
! Initialize variables                                                   -
!                                                                        -
!-------------------------------------------------------------------------

Defaults       let            start_angle=1024
               let            exec_time=210

!-------------------------------------------------------------------------
!                                                                        -
! Program Setup                                                          -
!  Configure map.                                                        -
!  Prepare piecewise profile.                                            -
!  Define and enable software interrupts.                                -
!                                                                        -
!-------------------------------------------------------------------------

Setup          no_op

               gosub          Set_Config
               gosub          Calc_Profile

               swi_if_on      0,FEED_DOWN,Fault
               enable_swi

!-------------------------------------------------------------------------
!                                                                        -
! Main body of program                                                   -
!  Enable FEED.                                                          -
!  Remain in Loop, waiting for transition on EXEC_PROF to execute        -
!    profile.                                                            -
!                                                                        -
!-------------------------------------------------------------------------

Main           turn_on        SYS_OK
               drive_on       FEED
               delta_comp     FEED,1,1,30,0,1,6000

Loop           if_io_on       EXEC_PROF,Loop
Loop1          if_io_off      EXEC_PROF,Loop1
               if_io_on       EXEC_TYPE,With_Lock

!-------------------------------------------------------------------------
!                                                                        -
! Routine to execute a piecewise profile.                                -
!  Execute the piecewise profile.                                        -
!  Measure the time to complete the profile, and compare to calculated   -
!    time.  If not correct stop system.                                  -
!  Otherwise return to Loop.                                             -
!                                                                        -
!-------------------------------------------------------------------------
               get_time       strt_time
               get_ms_time    strt_ms_tm
Profile        exec_profile   FEED
               gosub          Wait_Stop
               get_time       end_time
               get_ms_time    end_ms_tm
```

```
                let             diff=end_time-strt_time
                let             diff_ms=end_ms_tm-strt_ms_tm
                let             error=exec_time-diff
                let             temp=abs(error)
                if              temp<=2,Loop
                turn_off        SYS_OK
                sys_return


!-------------------------------------------------------------------------
!                                                                        -
! Routine to execute piecewise profile using lock mode 3.               -
!  Define the trigger angle and lock the FEED.                           -
!  Wait for motion to stop, and return to Loop.                          -
!                                                                        -
!-------------------------------------------------------------------------

With_Lock       set_trig_pw     FEED,start_angle
                lock            FEED,LK_MODE_3
                set_speed       MASTER,100
                set_ac_dc       MASTER,50
                jog_cw          MASTER
                gosub           Wait_Stop
                goto            Loop

!-------------------------------------------------------------------------
!                                                                        -
! Software Interrupt Routine used for a FEED fault condition.           -
!  Turn off the System Ok output.                                        -
!  Check to see that all motion has stopped.                             -
!  Wait for reset input, then restart at Main.                           -
!                                                                        -
!-------------------------------------------------------------------------

Fault           no_op
                turn_off        SYS_OK
                f_decel         FEED
                f_decel         MASTER


                set_tmr         72,200
Wait_Fault      if_tmr_on       72,Wait_Fault

Start_Over      if_io_off       RESET,Start_Over
                drive_off       FEED
                enable_swi
                restart_at      Main

!-------------------------------------------------------------------------
!                                                                        -
! Subroutine to configure the MAB map.                                  -
!  Set the appropriate bits, and check the results.                      -
!                                                                        -
!-------------------------------------------------------------------------
```

```
Set_Config      let             map=0
                set_map         map
                set_bit         MABA_TALKER,map
                set_bit         MABA_LISTEN,map
                set_map         map
                get_map         map_back
                get_map_stat    map_stat

                let             mcf=0
                set_mcf         MASTER,mcf

                set_bit         PSEUDO_MABA,mcf
                set_mcf         MASTER,mcf
                get_mcf         MASTER,mcf_back
                get_mcf_stat    MASTER,mcf_stat

                if              map_stat<>0,Config_Fail
                if              map<>map_back,Config_Fail
                if              mcf_stat<>0,Config_Fail
                if              mcf<>mcf_back,Config_Fail

Exit_Config     return_sub

Config_Fail     sys_return

!-------------------------------------------------------------------------
!                                                                        -
! Subroutine used to prepare the piecewise profile.                      -
!  Prepare the piecewise profile.                                        -
!  Check the profile status. If correct return, otherwise stop program.  -
!                                                                        -
!-------------------------------------------------------------------------

Calc_Profile    prep_profile    FEED,prodata


Calc_Busy       if_stat_on      FEED_CALC,Calc_Busy

                get_pstat       FEED,prof_stat
                if              prof_stat=0,Exit_Calc
                sys_fault

Exit_Calc       return_sub

!-------------------------------------------------------------------------
!                                                                        -
! Subroutine used to create a time delay.                                -
!  Wait for timer to time out and return.                                -
!                                                                        -
!-------------------------------------------------------------------------

Delay           set_tmr         TIMER,time
Wait_Time       if_tmr_on       TIMER,Wait_Time
                return_sub
```

```
!-------------------------------------------------------------------------
!                                                                        -
! Subroutine to wait for motion to be completed.                         -
!                                                                        -
!-------------------------------------------------------------------------

Wait_Stop      no_op
               if_stat_on     FEED_BUSY,Wait_Stop  ! check for axis busy
               return_sub
```

Ratio Lock

```
!-------------------------------------------------------------------------
! TITLE:       Ratio Lock
!
! DATE:        12/8/97
!
! DESCRIPTION: This program uses the fiber optic input channel as the
!              master data.  The lock is a simple ratio lock, meaning
!              that the slave will track the master proportional to the
!              currently set ratio.  With the use of the Analyzer, the
!              values of the ratio can be changed on the fly and when
!              the CHG_RATIO input is triggered, the new ratio will take
!              effect.
!
! EQUIPMENT:   DeltaMax
!
! COMMANDS: delta_comp          digi_comp         get_act_spd
!           get_fol_err         get_for_ang       get_for_spd
!           ratio
!
! The above commands are commonly used in many programs.  This
! illustration provides the context and syntax required to successfuly
! integrate these commands into a complete program.  While the use of
! these commands varies from program to program, the following example
! will give the programmer an indication of a typical usage.
!-------------------------------------------------------------------------

              declare        on

!-------------------------------------------------------------------------
!                                                                        -
! Define constants and variables                                         -
!                                                                        -
!-------------------------------------------------------------------------

!
!  Axis related constants
!

SHEER          equ            1
FIBER          equ            2

BUSY           equ            78
DOWN           equ            77
SHEER_BUSY     equ            (SHEER*16)+BUSY

SHEER_DOWN     equ            (SHEER*16)+DOWN


LK_MODE_1      equ            1
FIBER_MABA     equ            8              ! FIBER sends data to MABA
MABA_TALKER    equ            30             ! FIBER is talker on MABA
MABA_LISTEN    equ            23             ! SHEER is listener on MABA

!
```

```
!  I/O and Flags
!

CHG_RATIO      equ            0              ! Input to change ratio.
RESET          equ            6              ! Input to reset a SHEER
!                                               fault.

SYS_OK         equ            16             ! Output indicating system
!                                               ok.

TIMER          equ            72             ! Timer used to create
!                                               delays.
VG_CHK         equ            255            ! Flag indicating vgain
!                                               set.

!
!  Variable definitions
!

time           integer        ! Determines the delay time for a timer.

map            integer        ! Used to configure the MASTER/SLAVE on
!                               the MAB.
map_back       integer        ! Return variable for the current 'map'
!                               setup.
map_stat       integer        ! Return variable for the current 'map'
!                               status.
mcf            integer        ! Used to configure the FIBER.
mcf_back       integer        ! Return variable for the current 'mcf'
!                               setup.

mcf_stat       integer        ! Return variable for the current 'mcf'
!                               status.
ratio          integer        ! The current master/slave ratio.
aj2            integer        ! The load inertia mismatch.
aj3            integer        ! The high frequency response.
aj4            integer        ! The position gain in the system.
aj7            integer        ! The zero speed gain of the system.
aj8            integer        ! The velocity feed forward.
aj9            integer        ! The notch filter.
angle          integer        ! Stores the current fiber optic angle.
angle2         integer        ! Stores the old fiber optic angle.
error          integer        ! Stores the amount of following error.
temp           integer        ! Used as a temporary storage location.
scale          integer        ! The scale factor for the ratio.

sheer_spd      integer        ! The current SHEER speed.
cur_f_spd      integer        ! The current fiber optic speed.
new_f_spd      integer        ! The fiber optic speed scaled by the
!                               ratio.

!-------------------------------------------------------------------------
!                                                                        -
! Initialize variables                                                   -
!                                                                        -
!-------------------------------------------------------------------------
```

```
Defaults        let             ratio=4096
                let             time=1
                let             aj2=1
                let             aj3=1
                let             aj4=30
                let             aj7=0
                let             aj8=1
                let             aj9=6000

!-------------------------------------------------------------------------
!                                                                        -
! Program Setup                                                          -
!  Define axis motion parameters.                                        -
!  Configure map.                                                        -
!  Wait for motion on fiber optic channel.                               -
!  Define software interrupts.                                           -
!                                                                        -
!-------------------------------------------------------------------------


Setup           set_ac_dc       SHEER,400

                gosub           Set_Config

                get_for_ang     FIBER,1,angle

Wait_Angle      no_op
                get_for_ang     FIBER,1,angle2
                if              angle2=angle,Wait_Angle

                swi_if_on       0,SHEER_DOWN,Fault
                swi_if_on       1,CHG_RATIO,New_Ratio
                enable_swi

!-------------------------------------------------------------------------
!                                                                        -
! Main body of program                                                   -
!  Enable SHEER.                                                         -
!  Get information about current speeds.                                 -
!  Wait until speeds are the same, then check for vgain.                 -
!    If vgain has been set continue in Loop, otherwise calculate vgain.  -
!                                                                        -
!-------------------------------------------------------------------------

Main            turn_on         SYS_OK
                drive_on        SHEER
                digi_comp       1,32,0,0
                ratio           SHEER,ratio
                lock            SHEER,LK_MODE_1
                let             scale=ratio*100
                let             scale=scale/4096

Loop            get_act_spd     SHEER,sheer_spd
                get_for_spd     FIBER,1,cur_f_spd
                get_fol_err     SHEER,error
                let             new_f_spd=cur_f_spd*scale

                let             new_f_spd=new_f_spd/100
                set_tmr         TIMER,time
```

```
Delay           if_tmr_on       TIMER,Delay
                if              sheer_spd<>new_f_spd,Loop
                if_flag_on      VG_CHK,Loop
                delta_comp      SHEER,aj2,aj3,aj4,aj7,aj8,aj9
                goto            Loop

!-------------------------------------------------------------------------
!                                                                        -
! Software Interrupt Routine used for a SHEER fault condition.           -
!  Turn off the System Ok output.                                        -
!  Check to see that all motion has stopped.                             -
!  Wait for reset input, then restart at Main.                           -
!                                                                        -
!-------------------------------------------------------------------------

Fault           no_op
                turn_off        SYS_OK

                f_decel         SHEER

                set_tmr         TIMER,200
Wait_Fault      if_tmr_on       TIMER,Wait_Fault

Start_Over      if_io_off       RESET,Start_Over
                drive_off       SHEER
                enable_swi
                restart_at      Main

!-------------------------------------------------------------------------
!                                                                        -
! Subroutine to configure the MAB map and the MCF map.                   -
!  Set the appropriate bits, and check the results.                      -
!                                                                        -
!-------------------------------------------------------------------------

Set_Config      let             map=0
                set_map         map
                set_bit         MABA_TALKER,map
                set_bit         MABA_LISTEN,map
                set_map         map
                get_map         map_back
                get_map_stat    map_stat

                if              map_stat<>0,Config_Fail
                if              map<>map_back,Config_Fail

Config_Pseudo   let             mcf=0
                set_mcf         FIBER,mcf

                set_bit         FIBER_MABA,mcf
                set_mcf         FIBER,mcf
                get_mcf         FIBER,mcf_back
                get_mcf_stat    FIBER,mcf_stat

                if              mcf_stat<>0,Config_Fail
                if              mcf<>mcf_back,Config_Fail

Exit_Config     return_sub

Config_Fail     sys_return
```

```
!-------------------------------------------------------------------------
!                                                                        -
! Software Interrupt Routine used to change curretn ratio.              -
!  Implement new ratio and scaling.                                      -
!  Enable interrupts and return.                                         -
!                                                                        -
!-------------------------------------------------------------------------

New_Ratio      ratio          SHEER,ratio
               let            scale=ratio*100
               let            scale=scale/4096
               enable_swi
               return_sub
```

CAMS with calc_unit_cam

```
!-------------------------------------------------------------------------
! TITLE:       CAMS with calc_unit_cam
!
! DATE:        12/3/97
!
! DESCRIPTION: This program implements the 'calc_unit_cam' feature to
!              calculate a cam of specified length.  The lock method
!              demonstrated is mode 9, which causes the slave to lock
!              at a specific angle of the master.  This particular lock
!              method unlocks after completing one cam cycle.  In this
!              program the lock will be reinitiated each time the cam
!              cycle is completed.
!
! EQUIPMENT:   DeltaMax
!
! COMMANDS: begin_data          calc_cam_sum        calc_unit_cam
!           data                data_scale          dim
!           dim(card)           enable_swi          end_data
!           get_cam_sum         get_angle           let_byte
!           master_scale        set_trig_cam        swi_if_off
!           swi_if_on
!
! The above commands are commonly used in many programs.  This
! illustration provides the context and syntax required to successfuly
! integrate these commands into a complete program.  While the use of
! these commands varies from program to program, the following example
! will give the programmer an indication of a typical usage.
!-------------------------------------------------------------------------

              declare        on

!-------------------------------------------------------------------------
!                                                                        -
! Declare constants and variables                                        -
!                                                                        -
!-------------------------------------------------------------------------

!
!  Axis related constants
!

DRAW          equ            1
MASTER        equ            2

BUSY          equ            78              ! busy flag offset
DOWN          equ            77              ! down flag offset
CALC          equ            74              ! calculating flag offset
LOCK          equ            79              ! lock flag offset
DRAW_BUSY     equ            (DRAW*16)+BUSY  ! busy flag - DRAW
DRAW_DOWN     equ            (DRAW*16)+DOWN  ! down flag - DRAW
DRAW_CALC     equ            (DRAW*16)+CALC  ! calculating flag - DRAW
DRAW_LOCK     equ            (DRAW*16)+LOCK  ! lock flag - DRAW

MS            equ            6              ! 1/64
```

```
DS              equ             0               ! no multiply
LK_MODE_9       equ             9               ! lock mode 9
MABA_TALKER     equ             30              ! MASTER is talker on MABA
MABA_LISTEN     equ             23              ! DRAW is listener on MABA

!
!  I/O and Flags
!
SET_M_SCALE     equ             0               ! Change the master scale
SET_D_SCALE     equ             1               ! Change the data scale
RESET           equ             6               ! Reset Fault (input)
SYS_OK          equ             16              ! System okay (output)

TIMER           equ             72              ! Timer used for delays.

!
!  Variable Definitions
!

angle           integer         ! The angle of the master axis.
distance        integer         ! Defines the distance of the cam.
start_cam       integer         ! The position in memory to start the cam.
end_cam         integer         ! The ending position of the cam in memory.
num_elements    integer         ! The number of cam elements.
m_scale         integer         ! Defines the master scale.
d_scale         integer         ! Defines the data scale.

sum             integer         ! The sum of all the cam elements.
start_angle     integer         ! The start angle for the Lock to begin.
tmp             integer         ! Used as a temporary storage variable.
ctr             integer         ! Used as a loop counter.
ctr2            integer         ! Used as a loop counter.
time            integer         ! Determines the delay time for a timer.
time2           integer         ! Determines the delay time for a timer.
map             integer         ! Used to configure the MASTER/SLAVE on
!                                 the MAB.
map_back        integer         ! Return variable for the current 'map'
!                                 setup.
map_stat        integer         ! Return variable for the current 'map'
!                                 status.
mcf             integer
!
!  Data arrays and unit cam table
!

cam             dim             DRAW,6750
table           dim             DRAW,130
copy            dim             300

trap            begin_data
                data            0,14,87,205,357
                data            568,744,976,1227,1496
                data            1789,2097,2422,2761,3117
                data            3487,3671,4267,4677,5100
                data            5535,5882,6440,6909,7390
                data            7661,8380,8890,9410,9940
```

```
                data            10480,11030,11588,12153,12728
                data            13310,13902,14501,15109,15725
                data            16349,16961,17620,18266,18919
                data            19579,20246,20919,21596,22281
                data            22968,23659,24355,25052,25753
                data            26454,27155,27856,28557,29258
                data            29959,30660,31361,32062,32763
                data            33464,34165,34866,35567,36288
                data            36969,37670,38371,39072,39773
                data            40474,41171,41867,42560,43247
                data            43930,44609,45282,45949,46609
                data            47262,47908,48547,49179,49803
                data            50419,51027,51626,52218,52800
                data            53375,53940,54498,55048,55586
                data            56118,56638,57148,57647,58138
                data            58619,59088,59546,59993,60428
                data            60851,61261,61657,62041,62441
                data            62767,63106,63431,63739,64030
                data            64301,64552,64784,64990,65171
                data            65323,65441,65514,65536,65536
                end_data

!-------------------------------------------------------------------------
!                                                                        -
! Initialize variables                                                   -
!                                                                        -
!-------------------------------------------------------------------------

Defaults        let             distance=16384
                let             num_elements=300
                let             start_cam=0
                let             end_cam=num_elements-1
                let             start_angle=1024
                let             m_scale=MS
                let             d_scale=DS
!-------------------------------------------------------------------------
!                                                                        -
! Program Setup                                                          -
!  Setup the unit cam table.                                             -
!  Configure map.                                                        -
!  Define and enable software interrupts.                                -
!                                                                        -
!-------------------------------------------------------------------------

Setup           no_op

                gosub           Setup_Unit_Cam
                gosub           Set_Config

                swi_if_on       0,DRAW_DOWN,Fault
                swi_if_off      1,DRAW_LOCK,Lock_Ang
                swi_if_on       2,SET_M_SCALE,New_m_scale
                swi_if_on       3,SET_D_SCALE,New_d_scale
                enable_swi
```

```
!-------------------------------------------------------------------------
!                                                                        -
! Main body of program                                                   -
!  Enable DRAW.                                                          -
!  Setup DRAW for lock in mode 9.                                        -
!  Get status for DRAW.                                                  -
!  Remain in Loop, waiting for software interrupts.                      -
!                                                                        -
!-------------------------------------------------------------------------
Main            turn_on         SYS_OK
                drive_on        DRAW
                delta_comp      DRAW,1.0,1.0,30,0,1.0,6000   ! default values
                cam_data        DRAW,cam,MS,DS

                set_trig_cam    DRAW,start_angle
                lock            DRAW,LK_MODE_9

Loop            no_op
                get_angle       DRAW,angle
                goto            Loop

!-------------------------------------------------------------------------
!                                                                        -
! Software Interrupt Routine indicating that the cam has completed      -
!   and the DRAW has been unlocked.                                      -
!  Lock the DRAW axis in mode 9.                                         -
!  Enable the software interrupts and return.                            -
!                                                                        -
!-------------------------------------------------------------------------

Lock_Ang        no_op
                set_trig_cam    DRAW,start_angle
                lock            DRAW,LK_MODE_9
                enable_swi
                return_sub

!-------------------------------------------------------------------------
!                                                                        -
! Software Interrupt Routine used for a DRAW fault condition.           -
!  Turn off the System Ok output.                                        -
!  Check to see that all motion has stopped.                             -
!  Wait for reset input, then restart at Main.                           -
!                                                                        -
!-------------------------------------------------------------------------

Fault           no_op
                turn_off        SYS_OK
                f_decel         DRAW

                set_tmr         TIMER,200
Wait_Fault      if_tmr_on       TIMER,Wait_Fault

Start_Over      if_io_off       RESET,Start_Over
                drive_off       DRAW
                enable_swi
                restart_at      Main
```

```
!-------------------------------------------------------------------------
!                                                                        -
! Subroutine to configure the MAB map.                                   -
!  Set the appropriate bits, and check the results.                      -
!                                                                        -
!-------------------------------------------------------------------------

Set_Config     let            map=0
               set_map        map
               set_bit        MABA_TALKER,map
               set_bit        MABA_LISTEN,map
               set_map        map
               get_map        map_back
               get_map_stat   map_stat

               let            mcf=0
               set_mcf        MASTER,mcf
               set_bit        8,mcf
               set_mcf        MASTER,mcf

               if             map_stat<>0,Config_Fail
               if             map<>map_back,Config_Fail

Exit_Config    return_sub

Config_Fail    sys_return

!-------------------------------------------------------------------------
!                                                                       -
! Subroutine to prepare and create the unit cam.                        -
!  Clear the contents of the cam array.                                 -
!  Load the unit cam array into memory on DRAW.                         -
!  Calculate the cam based on the distance, starting location, and      -
!    number of elements.                                                -
!  Calculate the cam sum and compare it to the desired distance.        -
!    If the sum does not equal the distance, then stop system.          -
!  To observe the cam elements they are loaded into memory on the MSC.  -
!                                                                       -
!-------------------------------------------------------------------------

Setup_Unit_Cam no_op
               let            ctr=0
Clr_Cam        let            cam[ctr]=0
               let            ctr=ctr+1
               if             ctr<6750,Clr_Cam

Load_Table     let            ctr=0
Load_Loop      let            tmp=trap[ctr]
               let            table[ctr]=tmp
               let            ctr=ctr+1
               if             ctr<130,Load_Loop

               calc_unit_cam  DRAW,distance,num_elements,start_cam
Calc_Busy      if_stat_on     DRAW_CALC,Calc_Busy

               calc_cam_sum   DRAW,start_cam,end_cam
```

```
Sum_Busy        if_stat_on      DRAW_CALC,Sum_Busy

                get_cam_sum     DRAW,sum
                if              sum=distance,Copy

Calc_Err        sys_fault

Copy            let             ctr=0
Load_Copy       let_byte        tmp=cam[ctr]
                let_byte        copy[ctr]=tmp
                let             ctr=ctr+1
                if              ctr<300,Load_Copy
                return_sub

!-------------------------------------------------------------------------
!                                                                        -
! Subroutine to modify the master scale of the cam                       -
!  Each time the routine is executed the master scale will toggle.       -
!                                                                        -
!-------------------------------------------------------------------------

New_m_scale     no_op
                if              m_scale=6,Change_5
                let             m_scale=6
                master_scale    DRAW,m_scale
                goto            Ex_m_scale

Change_5        let             m_scale=5
                master_scale    DRAW,m_scale

Ex_m_scale      enable_swi
                return_sub

!-------------------------------------------------------------------------
!                                                                        -
! Subroutine to modify the data scale of the cam                         -
!  Each time the routine is executed the data scale will toggle.         -
!                                                                        -
!-------------------------------------------------------------------------

New_d_scale     no_op
                if              d_scale=0,Change_1
                let             d_scale=0
                data_scale      DRAW,d_scale
                goto            Ex_d_scale

Change_1        let             d_scale=1
                data_scale      DRAW,d_scale

Ex_d_scale      enable_swi
                return_sub
```

I/O Functions

```
!-------------------------------------------------------------------------
! TITLE:       I/O functions
!              bus to select a particular type of motion.
!
! DATE:        12/5/97
!
! EQUIPMENT:   Hardware is wired in the following manner;
!              DeltaMax
!
!                Outputs  Inputs
!                  IO#16 -> IO#0
!                  IO#17 -> IO#1
!                  IO#18 -> IO#2
!
! COMMANDS: blk_io_in          blk_io_out      if_tmr_off
!           vel_ccw            vel_cw
!
! The above commands are commonly used in many programs.  This
! illustration provides the context and syntax required to successfuly
! integrate these commands into a complete program.  While the use of
! these commands varies from program to program, the following example
! will give the programmer an indication of a typical usage.
!-------------------------------------------------------------------------

               declare        on

!-------------------------------------------------------------------------
!                                                                        -
!   Declare constants and variable                                       -
!                                                                        -
!-------------------------------------------------------------------------

!
!   Axis related constants
!

ROLL_1         equ            1

BUSY           equ            78
DOWN           equ            77
ROLL_1_BUSY    equ            (ROLL_1*16)+BUSY

ROLL_1_DOWN    equ            (ROLL_1*16)+DOWN

!
!   I/O and Flags
!

BLK_OUT        equ            16              ! start of output IO block
!                                                (8 IO's)
BLK_IN         equ            0               ! start of input IO block
!                                                (8 IO's)
RESET          equ            8               ! Reset Fault (input)
TIMER          equ            72              ! Timer Flag
```

```
!
!   Variable Definition
!

code            integer                     ! output variable to IO
!                                             block
copy            integer                     ! input variable to IO
!                                             block
ac_dc           integer                     ! acceleration,deceleration
!                                             ratio
speed           integer                     ! speed
time_delay      integer                     ! variable carrying
!                                             required delay
ctr             integer                     ! array indexer (counter)

!
!   Array Definition
!

c_array         begin_data                  ! array with codes (binary)
                data            1,2,3,4
                end_data
!-------------------------------------------------------------------------
!                                                                        -
!   Initialize variables                                                 -
!                                                                        -
!-------------------------------------------------------------------------

Defaults        no_op
                let             ac_dc=89
                let             speed=200
                let             ctr=0

!-------------------------------------------------------------------------
!                                                                        -
!   Program Setup                                                        -
!  Define ROLL_1 motion parameters.                          -
!  Define and enable software interrupt.                               -
!                                                                        -
!-------------------------------------------------------------------------
Setup           no_op
                set_ac_dc       ROLL_1,ac_dc    ! set accel. decel. for
!                                                 roll 1
                set_speed       ROLL_1,speed    ! set speed for roll 1

                swi_if_on       0,ROLL_1_DOWN,Fault
                enable_swi
```

```
!-------------------------------------------------------------------------
!                                                                        -
!           Main body of the program.                                    -
!           Send a binary code from array throgh the output IO's.        -
!           Read code into 'copy' through the 8 input IO's               -
!           Compare and direct to motion routine.                        -
!           Increment array counter, get next value and loop again.      -
!                                                                        -
!-------------------------------------------------------------------------
Main            no_op
                drive_on        ROLL_1

Start           no_op
                let             code=c_array[ctr]
                blk_io_out      BLK_OUT,code

                let             time_delay=0
                gosub           Delay

Block_Read      no_op
                let             time_delay=2000
                blk_io_in       BLK_IN,copy
                if              copy=1,Run_Roll1
                if              copy=2,Run_Roll2
                if              copy=3,Run_Roll1ccw
                if              copy=4,Run_Roll2ccw
                goto            Block_Read

Run_Roll1       no_op
                vel_cw          ROLL_1
                gosub           Delay
                f_decel         ROLL_1

Wait_Stop1      no_op
                if_stat_on      ROLL_1_BUSY,Wait_Stop1
                goto            Do_Next

Run_Roll2       no_op
                jog_cw          ROLL_1
                gosub           Delay
                f_decel         ROLL_1

Wait_Stop2      no_op
                if_stat_on      ROLL_1_BUSY,Wait_Stop2
                goto            Do_Next

Run_Roll1ccw    no_op
                vel_ccw         ROLL_1
                gosub           Delay
                f_decel         ROLL_1

Wait_Stop1ccw   no_op
                if_stat_on      ROLL_1_BUSY,Wait_Stop1ccw
                goto            Do_Next

Run_Roll2ccw    no_op
                jog_ccw         ROLL_1
                gosub           Delay
                f_decel         ROLL_1
```

```
Wait_Stop2ccw  no_op
               if_stat_on     ROLL_1_BUSY,Wait_Stop2ccw
               goto           Do_Next

Do_Next        no_op
               let            ctr=ctr+1
               if             ctr>3,Program_End
               goto           Start

Program_End    no_op
               let            ctr=0
               goto           Start

!-------------------------------------------------------------------------
!                                                                        -
!  Subroutine used to create a time delay.                               -
!   Wait until the timer times out.                                      -
!                                                                        -
!-------------------------------------------------------------------------
Delay          no_op
               set_tmr        TIMER,time_delay
Wait_Tmr       if_tmr_off     TIMER,Exit_Delay
               goto           Wait_Tmr

Exit_Delay     no_op
               return_sub

!-------------------------------------------------------------------------
!                                                                        -
! Software Interrupt Routine used for a ROLL_x fault condition.          -
!  Turn off the System Ok output.                                        -
!  Check to see that all motion has stopped.                             -
!  Reset 'ctr' and place controller in test mode.                        -
!  Wait for reset input, then restart at Main.                           -
!                                                                        -
!-------------------------------------------------------------------------

Fault          no_op
               f_decel        ROLL_1

               set_tmr        TIMER,200
Wait_Fault     if_tmr_on      TIMER,Wait_Fault

Start_Over     if_io_off      RESET,Start_Over
               drive_off      ROLL_1
               let            ctr=0
               enable_swi
               restart_at     Main
```

Offset Methods

```
!-------------------------------------------------------------------------
! TITLE:       Offset methods
!
! DATE:        12/8/97
!
! DESCRIPTION: This program uses MASTER-SLAVE ratios to demonstrate
!              the offset and speed control of a master axis.
!
! EQUIPMENT:   DeltaMax
!
! COMMANDS: incr_com            offset_master       read_offset
!           set_offset
!
! The above commands are commonly used in many programs.  This
! illustration provides the context and syntax required to successfuly
! integrate these commands into a complete program.  While the use of
! these commands varies from program to program, the following example
! will give the programmer an indication of a typical usage.
!-------------------------------------------------------------------------

               declare        on


!-------------------------------------------------------------------------
!                                                                        -
!  Declare constants and variables                                       -
!                                                                        -
!-------------------------------------------------------------------------


!
!  Axis related constants
!

CONVEYOR       equ            1
PSEUDO         equ            2

BUSY           equ            78
DOWN           equ            77
CONVEYOR_BUSY  equ            (CONVEYOR*16)+BUSY
PSEUDO_BUSY    equ            (PSEUDO*16)+BUSY

CONVEYOR_DOWN  equ            (CONVEYOR*16)+DOWN

MABA_TALKER    equ            30                    ! PSEUDO is talker
!                                                      on MABA
MABA_LISTEN1   equ            23                    ! CONVEYOR listen
!                                                      to MABA
PSEUDO_MABA    equ            10                    ! PSEUDO talks to
!                                                      MABA

!
!  I/O and Flags
!
```

```
INCREMENT       equ             0               ! Activate Inc. Conv.Pos.
!                                                 (input)
ADVANCE         equ             1               ! Activate offset master
!                                                 (input)
POSITIONING     equ             2               ! Activate positioning
!                                                 (input)
RESET           equ             6               ! Reset Fault (input)
SYS_OK          equ             16              ! System okay (output)
TIMER           equ             72              ! Timer used for delays.
WATCHDOG        equ             73              ! Timer used for watchdog

!
!  Data tables
!

POS_ARRAY       begin_data
                data            10240,-40960,20480,0,40960,0

                end_data

!
!  Misc. Constants
!
WATCHDOG_TIME   equ             50              !     mSec

!
!  Variable definitions
!

map             integer                         ! Var for configuring
!                                                 Hardware
map_copy        integer                         ! Var used to verify config.
map_stat        integer                         ! map_stat = 0 if map OK,
!                                                 <> 0 if error.
mcf             integer                         ! Var used to setup MCF
mcf_copy        integer                         ! Var used to verify MCF
!                                                 setup
mcf_stat        integer                         ! mcf_stat = 0 if mcf OK,
!                                                 <> 0 if error.
time_delay      integer                         ! Var with delay requirement
ac_dc           integer                         ! Accel decel
speed           integer                         ! Speed
offset          integer                         ! Var used to offset master
offset_copy     integer                         ! Angle read back from
!                                                 offset
conv_offset     integer                         ! Var used to inc. pos
!                                                 for Conv.
advance         integer                         ! Var used to move master
!                                                 offset
advance_copy    integer                         ! Used to verify move by
!                                                 advance
ctr             integer                         ! counter, index of pos
!                                                 array
actual_pos      integer                         ! Var containing value
!                                                 from array
```

```
time_mark      integer                      ! Test variable used for
!                                             time test
delta_time     integer                      ! Var used to calc.
!                                             interval
start_pos      integer                      ! Var used to get starting
!                                             pos
conv_pos       integer                      ! Var used to get commanded
!                                             pos
delta_pos      integer                      ! Var used to measure
!                                             displacement
no_of_times    integer                      ! Software trap for one
!                                             time usage
millisec       integer                      ! Motor interrupts in mSec.


!-------------------------------------------------------------------------
!                                                                        -
!  Initialize variables                                                  -
!                                                                        -
!-------------------------------------------------------------------------

Defaults       no_op
               let            ac_dc=300
               let            speed=1000
               let            offset=1024
               let            conv_offset=1024
               let            advance=2048
               let            millisec=200
               let            ctr=0
               let            no_of_times=0
               let            time_mark=0
               let            conv_pos=0
               let            start_pos=0



!-------------------------------------------------------------------------
!                                                                        -
!  Program setup                                                         -
!  Define motion parameters.                                             -
!  Configure map.                                                        -
!  Define software interrupts.                                           -
!                                                                        -
!-------------------------------------------------------------------------

Setup          no_op
               set_ac_dc      PSEUDO,ac_dc
               set_speed      PSEUDO,speed
               set_ac_dc      CONVEYOR,ac_dc
               set_speed      CONVEYOR,speed

               gosub          Set_Config

               swi_if_on      0,CONVEYOR_DOWN,Fault
               swi_if_on      2,INCREMENT,Inc_Conveyor
               swi_if_on      3,ADVANCE,Set_Advance
               swi_if_on      4,POSITIONING,Positioning
               enable_swi
```

```
!-------------------------------------------------------------------------
!                                                                        -
!  Main body of the program.                                             -
!   Start timed motion.                                                  -
!   Start Position of Master.                                            -
!                                                                        -
!-------------------------------------------------------------------------

Main           no_op
               turn_on         SYS_OK
               drive_on        CONVEYOR
               delta_comp      CONVEYOR,1.0,1.0,30,0,1.0,6000
Stay           goto            Stay



!-------------------------------------------------------------------------
!                                                                        -
!  Increment Offset Conveyor                                             -
!   Test for proper displacement.                                        -
!                                                                        -
!-------------------------------------------------------------------------

Inc_Conveyor   no_op

               let             conv_pos=0
               let             start_pos=0

               get_com         CONVEYOR,start_pos
               incr_com        CONVEYOR,conv_offset,millisec

               set_tmr         WATCHDOG,WATCHDOG_TIME
Get_Inc_Pos    get_com         CONVEYOR,conv_pos
               let             delta_pos = conv_pos - start_pos
               if_tmr_off      WATCHDOG,Fail_Inc
               if              delta_pos<>conv_offset,Get_Inc_Pos


               enable_swi
               return_sub

Fail_Inc       no_op
               gosub           Invalid_Ofst


!-------------------------------------------------------------------------
!                                                                        -
!  Set Advance (Correction) for Master                                   -
!   Only to be used once per operation.                                  -
!   Note:Cycle power before running to clear angle position.             -
!                                                                        -
!-------------------------------------------------------------------------
```

```
Set_Advance    no_op
               if              no_of_times<>0,Exit_Advance
               let             no_of_times=1
               lock            CONVEYOR,1

               let             time_delay=20
               gosub           Delay

               offset_master   PSEUDO,advance

               let             time_delay=20
               gosub           Delay

               get_angle       CONVEYOR,advance_copy
               if              advance_copy<>advance,Invalid_Ofst


!
!  Move to Offset Location
!

               set_offset      PSEUDO,offset
Wait_Ready     if_stat_on      PSEUDO_BUSY,Wait_Ready
               read_offset     PSEUDO,offset_copy
               if              offset<>offset_copy,Invalid_Ofst

               unlock          CONVEYOR,0
Exit_Advance   enable_swi
               return_sub



!-------------------------------------------------------------------------
!                                                                        -
!  Positioning of master (and Slaves) with corrections (offsets)         -
!                                                                        -
!-------------------------------------------------------------------------

Positioning    no_op
               lock            CONVEYOR,1
               let             actual_pos=POS_ARRAY[ctr]
               position        PSEUDO,actual_pos
Wait_Position  if_stat_on      PSEUDO_BUSY,Wait_Position
               let             ctr=ctr+1
               if              ctr<6,Return_Main

!
!  End of routine
!
               let             ctr=0

Return_Main    unlock          CONVEYOR,0
               enable_swi
               return_sub
```

```
!-------------------------------------------------------------------------
!                                                                        -
!  Subroutine used to configure the MAB map and the MCF map.             -
!   Set the appropriate bits, and check the results.                     -
!                                                                        -
!-------------------------------------------------------------------------


Set_Config      let             map=0
                set_map         map
                set_bit         MABA_TALKER,map
                set_bit         MABA_LISTEN1,map
                set_map         map
                let             time_delay=30
                gosub           Delay

                get_map         map_copy
                get_map_stat    map_stat

                if              map<>map_copy,Config_Fail
                if              map_stat<>0,Config_Fail


Config_Pseudo   no_op
                let             mcf=0
                set_mcf         PSEUDO,mcf
                set_bit         PSEUDO_MABA,mcf
                set_mcf         PSEUDO,mcf

                let             time_delay=30
                gosub           Delay

                get_mcf         PSEUDO,mcf_copy
                get_mcf_stat    PSEUDO,mcf_stat

                if              mcf<>mcf_copy,Config_Fail
                if              mcf_stat<>0,Config_Fail


                enable_swi
                return_sub

Config_Fail     no_op
                sys_return



!-------------------------------------------------------------------------
!                                                                        -
!  Subroutine used to create a time delay.                               -
!   Wait until the timer times out.                                      -
!                                                                        -
!-------------------------------------------------------------------------
```

```
Delay           no_op
                set_tmr         TIMER,time_delay
Wait_Time       if_tmr_on       TIMER,Wait_Time
                return_sub


!-------------------------------------------------------------------------
!                                                                        -
!  Subroutine used to halt operation on erroneous offset                 -
!                                                                        -
!-------------------------------------------------------------------------

Invalid_Ofst    no_op
                f_decel         PSEUDO
Wait_Stop       if_stat_on      PSEUDO_BUSY,Wait_Stop
                f_decel         CONVEYOR
Wait_Loop       no_op
                if_stat_on      CONVEYOR_DOWN,Exit_Ofst
                if_stat_on      CONVEYOR_BUSY,Wait_Loop

Exit_Ofst       sys_return


!-------------------------------------------------------------------------
!                                                                        -
! Software Interrupt Routine used to handle fault conditions .           -
!  Turn off the System Ok output.                                        -
!  Check to see that all motion has stopped.                             -
!  Wait for reset input, then restart at Main.                           -
!                                                                        -
!-------------------------------------------------------------------------

Fault           no_op
                turn_off        SYS_OK
                f_decel         CONVEYOR

                set_tmr         TIMER,200
Wait_Fault      if_tmr_on       TIMER,Wait_Fault

Start_Over      if_io_off       RESET,Start_Over
                drive_off       CONVEYOR
                enable_swi
                restart_at      Main
```

Clear Software Interrupts

```
!-------------------------------------------------------------------------
! TITLE:       Clear Software Interrupts
!
! DATE:        12/3/97
!
! DESCRIPTION: This program was written to demonstrate the usage of
!              interrupt managing and predefined-function hardware
!              interrupt usage (trap_pos).
!
! EQUIPMENT:   DeltaMax
!
! COMMANDS:    clr_all_swi         clr_swi          set_ms_tmr
!              trap_pos
!
! The above commands are commonly used in many programs.  This
! illustration provides the context and syntax required to successfuly
! integrate these commands into a complete program.  While the use of
! these commands varies from program to program, the following example
! will give the programmer an indication of a typical usage.
!-------------------------------------------------------------------------

              declare        on

!-------------------------------------------------------------------------
!                                                                        -
!  Declare constants and variables                                       -
!                                                                        -
!-------------------------------------------------------------------------


!
!  Axis related constants
!

WHEEL          equ            1
BUSY           equ            78
DOWN           equ            77
WHEEL_BUSY     equ            (WHEEL*16)+BUSY
WHEEL_DOWN     equ            (WHEEL*16)+DOWN
WHEEL_IDX      equ            89
!
!  I/O and Flags
!

EMERGENCY      equ            6              ! Input
DIRECTION      equ            7              ! Input
TIMER          equ            72             ! Timer flag
HWI_ARMED_1    equ            91             ! Got position


!
!  Variable definitions
!
```

```
speed           integer
ac_dc           integer
time_delay      integer
next_bottle     integer
bottle          integer
pos_trapped     integer


!-------------------------------------------------------------------------
!                                                                        -
!  Initialize variables                                                  -
!                                                                        -
!-------------------------------------------------------------------------

Defaults        no_op
                let             ac_dc=50
                let             speed=1000
                let             bottle=40960
                let             next_bottle=40960
                let             time_delay=50

!-------------------------------------------------------------------------
!                                                                        -
!  Program setup                                                         -
!                                                                        -
!-------------------------------------------------------------------------

Setup           no_op
                gosub           Grab_Pos_1
                set_ac_dc       WHEEL,ac_dc
                set_speed       WHEEL,speed
                drive_on        WHEEL
                delta_comp      WHEEL,1.0,1.0,30,0,1.0,6000
                swi_if_on       0,EMERGENCY,Emergency
                swi_if_on       1,DIRECTION,Do_Forward
                swi_if_off      2,HWI_ARMED_1,Grab_Pos_1
                enable_swi


!-------------------------------------------------------------------------
!                                                                        -
!  Main body of the program.                                             -
!   This routine has only a loop to allow the interrupts to work         -
!                                                                        -
!-------------------------------------------------------------------------

Main            no_op
                goto            Main


!-------------------------------------------------------------------------
!                                                                        -
!  This routine was created to do a forward motion based on the level    -
!   of the IO (7) as s. interrupt signal.                                -
!                                                                        -
!-------------------------------------------------------------------------
```

```
Do_Forward      no_op
                let             next_bottle=bottle
                clr_swi         1
                swi_if_on       1,DIRECTION,Do_Reverse
                enable_swi
Idx_Loop        no_op
                index           WHEEL,next_bottle
Wait_Idx        if_stat_on      WHEEL_IDX,Wait_Idx

                goto            Idx_Loop

                return_sub


!-------------------------------------------------------------------------
!                                                                        -
!  This routine was created to do a reverse motion based on the level   -
!   of the IO (7) as s. interrupt signal.                                -
!                                                                        -
!-------------------------------------------------------------------------

Do_Reverse      no_op
                clr_swi         1
                swi_if_on       1,DIRECTION,Do_Forward
                enable_swi
Idx_Loop_R      no_op
                let             next_bottle=0-bottle
                index           WHEEL,next_bottle
Wait_Idx_R      if_stat_on      WHEEL_IDX,Wait_Idx_R

                goto            Idx_Loop_R

                return_sub



!-------------------------------------------------------------------------
!                                                                        -
!  Subroutine used to create a time delay of 50ms.                                -
!   Wait until the timer times out.                                      -
!                                                                        -
!-------------------------------------------------------------------------

Delay           no_op
                set_ms_tmr      TIMER,time_delay
Wait_Time       if_tmr_on       TIMER,Wait_Time
                return_sub


!-------------------------------------------------------------------------
!                                                                        -
!  Routine created to capture the present position of the motor when a  -
!  hardware interrupt comes in.                                         -
!                                                                        -
!-------------------------------------------------------------------------
```

```
Grab_Pos_1      no_op
                get_trap_pos    WHEEL,pos_trapped

                enable_hwi
                trap_pos        WHEEL

                enable_swi
                return_sub


!-------------------------------------------------------------------------
!                                                                        -
!  Subroutine used to service emergency stop.                            -
!                                                                        -
!-------------------------------------------------------------------------

Emergency       no_op
                clr_all_swi

                f_decel         WHEEL
Wt_Emrg_Stop    if_stat_on      WHEEL_DOWN,End_Emergency
                if_stat_on      WHEEL_BUSY,Wt_Emrg_Stop

End_Emergency   drive_off       WHEEL
                sys_return
```

Operator Interface

```
!-------------------------------------------------------------------------
! TITLE:        Operator Interface
!
! DATE:         12/8/97
!
! DESCRIPTION:  This program will prompt the operator to enter speed,
!               accel/decel rate and an absolute position
!               (as motor turns xx.xx) and then execute a move to that
!               position with the parameters specified.
!
!               As each entry is made, the program will verify that it is
!               within the proper range and will alert the operator if
     !              not.
!
! EQUIPMENT:    DeltaMax
!
! COMMANDS: case                default               end_select
!           exit_select         if_char               if_no_char
!           input               port_set              print
!           print_num           select                stop_input
!           text
!
! The above commands are commonly used in many programs.  This
! illustration provides the context and syntax required to successfuly
! integrate these commands into a complete program.  While the use of
! these commands varies from program to program, the following example
! will give the programmer an indication of a typical usage.
!-------------------------------------------------------------------------

              declare       off

AXIS_1        equ           1
DOWN_1        equ           93
BUSY_1        equ           94

TIMER_1       equ           72
BITS_TURN     equ           4096

DEF_SPEED     equ           10
MIN_SPEED     equ           1
MAX_SPEED     equ           2000

DEF_ACDC      equ           100
MIN_ACDC      equ           5
MAX_ACDC      equ           800

DEF_TURNS     equ           0
MIN_TURNS     equ           -9999         ! -99.99 turns
MAX_TURNS     equ           9999          ! 99.99 turns

PORT          equ           2

MINUS_1       equ           -1
ZERO          equ           48
```

```
F1              equ             65
F2              equ             66
F3              equ             67
F4              equ             68
NEXT            equ             78
PREV            equ             80
ENTER           equ             255

MENU_1_1        text            "F1 SETUP<13><10>"
MENU_1_2        text            "F2 RUN<13><10>"
MENU_1_3        text            "<13><10>"
MENU_1_4        text            "F4 STOP PROG<13><10>"

SPEED_IN        text            "SPEED:          "
ACDC_IN         text            "AC/DC:          "
TURNS_IN        text            "TURN:           "
POSITIONING     text            "POSITIONING ... "

ERR_1           text            "ENTRY:<13><10>"
ERR_2           text            "  OUT OF RANGE<13><10>"
ERR_3           text            "  TO<13><10>"
ERR_4           text            " PRESS ANY KEY  "

E1_TXT          text             "DRIVE FAULT"
E2_TXT          text             "POSITION ERROR"
E3_TXT          text             "PRESS F4 TO RESET"
NO_MATCH        text             "ERR-NO MATCH"

CRLF            text            "<13><10>"
NULL            text            ""
CURSOR_POS      text            "<27>I "
BACKLIGHT       text            "<27>V"
C_ON            text            "<27>bA"
C_OFF           text            "<27>b@"
C_HOME          text            "<27>H"
C_DOWN          text            "<27>B"
C_RIGHT         text            "<27>C"
CLR_SCREEN      text            "<27>E<27>H"

AUTO_WRAP       text            "<27>R@"
AUTO_SCROLL     text            "<27>S@"
AUTO_L_FEED     text            "<27>T@"
SET_CUR         text            "<27>bA"


CLR_TO_END      text            "<27>J"

!-------------------------------------------------------------------------
!
! -------- PROGRAM SETUP ---------
!
!-------------------------------------------------------------------------

                drive_on        AXIS_1
                set_gl_ccw      AXIS_1
```

```
               gosub          opi_setup
               let            speed=DEF_SPEED
               let            acdc=DEF_ACDC
               let            turns=DEF_TURNS

!-------------------------------------------------------------------------
!
! -------- MAIN MENU ---------
!
!-------------------------------------------------------------------------

main           print          CLR_SCREEN
               print          MENU_1_1
               print          MENU_1_2
               print          MENU_1_3
               print          MENU_1_4

main_key       gosub          ret_f_key
               if             key=F1,set_speed
               if             key=F2,run
               if             key=F4,exit
               goto           main_key

!-------------------------------------------------------------------------
!
! ---------- SETUP SPEED ----------
!
!-------------------------------------------------------------------------

set_speed      print          CLR_SCREEN
               print          SPEED_IN
               print_num      4,0,speed
               print          CRLF
               let            temp=speed

               input          NULL,7,0,temp,ENTER
speed_wt       if_flag_off    ENTER,speed_wt
               if             temp>MAX_SPEED,speed_err
               if             temp<MIN_SPEED,speed_err
               let            speed=temp
               goto           set_acdc

speed_err      gosub          err_screen
               let            row=0
               let            col=9
               gosub          cursor
               print_num      7,0,temp

               let            row=2
               let            col=0
               gosub          cursor
               print_num      4,0,MIN_SPEED

               let            row=2
               let            col=12
```

```
                gosub           cursor
                print_num       4,0,MAX_SPEED
                gosub           press_key
                goto            set_speed

!-------------------------------------------------------------------------
!
! ---------- SETUP ACCEL/DECEL RATE ----------
!
!-------------------------------------------------------------------------

set_acdc        print           CLR_SCREEN
                print           ACDC_IN
                print_num       4,0,acdc
                print           CRLF

                let             temp=acdc
                input           NULL,7,0,temp,ENTER
acdc_wt         if_flag_off     ENTER,acdc_wt
                if              temp>MAX_ACDC,acdc_err
                if              temp<MIN_ACDC,acdc_err
                let             acdc=temp
                goto            set_turns
acdc_err        gosub           err_screen
                let             row=0
                let             col=9
                gosub           cursor
                print_num       7,0,temp
                let             row=2
                let             col=0
                gosub           cursor
                print_num       3,0,MIN_ACDC

                let             row=2
                let             col=13
                gosub           cursor
                print_num       3,0,MAX_ACDC
                gosub           press_key
                goto            set_acdc

!-------------------------------------------------------------------------
!
! ---------- SETUP POSITION IN TURNS ----------
!
!-------------------------------------------------------------------------

set_turns       print           CLR_SCREEN
                print           TURNS_IN
                print_num       8,2,turns
                print           CRLF
                let             temp=turns

                input           NULL,8,2,temp,ENTER
turns_wt        if_flag_off     ENTER,turns_wt
                if              temp>MAX_TURNS,turns_err
                if              temp<MIN_TURNS,turns_err
                let             turns=temp
                goto            main
```

```
turns_err       gosub           err_screen
                let             row=0
                let             col=9
                gosub           cursor
                print_num       7,2,temp

                let             row=2
                let             col=0
                gosub           cursor
                print_num       6,2,MIN_TURNS

                let             row=2
                let             col=10
                gosub           cursor
                print_num       6,2,MAX_TURNS
                gosub           press_key
                goto            set_turns


!-------------------------------------------------------------------------
!
! -------- RUN MODE ---------
!
!-------------------------------------------------------------------------

run             print           CLR_SCREEN
                print           POSITIONING
                set_speed       AXIS_1,speed
                set_ac_dc       AXIS_1,acdc
                gosub           turns_to_bits

                position        AXIS_1,turn_bits
run_busy        if_stat_off     DOWN_1,r_b1
                let             error=1
                gosub           fault
                goto            main
r_b1            if_stat_on      BUSY_1,run_busy
                get_pos         AXIS_1,one_pos
                let             check_pos=turn_bits-one_pos
                let             check_pos=abs(check_pos)
                if              check_pos<5,main
                let             error=2
                gosub           fault
                goto            main

!-------------------------------------------------------------------------
!
! -------- CONVERT TURNS TO BITS --------
!
!-------------------------------------------------------------------------

turns_to_bits   let             turn_bits=0
                if              turns=0,ttb_end
                let             whole_turn=turns/100
                let             turn_bits=whole_turn*BITS_TURN
                let             whole_turn=whole_turn*100
                let             frac_turn=turns-whole_turn
                let             r_value=frac_turn*BITS_TURN
                gosub           round         ! intentionally 'round' twice
                gosub           round  ! due to turns to hundredths
                let             turn_bits=turn_bits+r_value
```

```
ttb_end         return_sub


!-------------------------------------------------------------------------
!
! -------- ROUND function --------
!
!-------------------------------------------------------------------------

round           let             result=r_value/10
                let             new_result=result*10
                let             diff=r_value-new_result
                let             r_value=result
                if              diff<5,end_round
                let             r_value=result+1
end_round       return_sub

!-------------------------------------------------------------------------
!
! -------- EXIT PROGRAM ---------
!
!-------------------------------------------------------------------------

exit            f_decel         AXIS_1
                let             time=200
                gosub           pause
                drive_off       AXIS_1
                print           CLR_SCREEN
                sys_return

!-------------------------------------------------------------------------
!
! -------- AXIS FAULT ENCOUNTERED ---------
!
!-------------------------------------------------------------------------

fault           f_decel         AXIS_1
                let             time=200
                gosub           pause
                gosub           display_err
                drive_off       AXIS_1
                gosub           pause
                drive_on        AXIS_1,1305
                goto            main
```

```
!-------------------------------------------------------------------------
!
! -------- OPI SETUP ---------
!
!-------------------------------------------------------------------------

opi_setup       port_set        2,9600,8
                let             time_1=10
                print           BACKLIGHT
                gosub           pause
                 print          AUTO_WRAP
                gosub           pause
                print           AUTO_L_FEED
                gosub           pause
                print           AUTO_SCROLL
                gosub           pause
                print           SET_CUR
                gosub           pause
op_exit         return_sub

!-------------------------------------------------------------------------
!
! -------- RETURNS THE SELECTED FUNCTION KEY ---------
!
!-------------------------------------------------------------------------

ret_f_key       print           C_OFF
                input           NULL,0,0,key,ENTER
rfk_wait        if_flag_off     ENTER,rfk_wait

                let             err=0
                if              key=F1,rfk_ok
                if              key=F2,rfk_ok
                if              key=F3,rfk_ok
                if              key=F4,rfk_ok
                if              key=NEXT,rfk_ok
                if              key=PREV,rfk_ok
                if              key=ENTER,rfk_ok
                goto            ret_f_key

rfk_ok          print           C_ON
                return_sub

!-------------------------------------------------------------------------
!
! -------- PRESS ANY KEY TO CONTINUE --------
!
!-------------------------------------------------------------------------

press_key       stop_input
                input           NULL,0,0,key,ENTER
pk_wait         if_flag_off     ENTER,pk_wait
                return_sub

!-------------------------------------------------------------------------
!
! -------- POSITION CURSOR FUNCTION ---------
!
!-------------------------------------------------------------------------
```

```
cursor          print           C_HOME

                let             ctr=0
c_next_col      if              ctr=col,c_row
                print           C_RIGHT
                let             ctr=ctr+1
                goto            c_next_col

c_row           let             ctr=0
c_next_row      if              ctr=row,c_done
                print           C_DOWN
                let             ctr=ctr+1
                goto            c_next_row
c_done          return_sub

!-------------------------------------------------------------------------
!
! -------- TEMPLATE FOR ERROR DISPLAY ---------
!
!-------------------------------------------------------------------------

err_screen      print           CLR_SCREEN
                print           ERR_1
                print           ERR_2
                print           ERR_3
                print           ERR_4
                return_sub

!-------------------------------------------------------------------------
!                                                                        -
! Subroutine to display error.                                           -
!  Use a select structure to determine the error to be displayed.        -
!  Print the error message and return.                                   -
!  To return the selected key.                                           -
!  Clear out the input buffer.                                           -
!  Wait for a key to be pressed.                                         -
!  Check if key is valid, and repeat if not valid or return.             -
!                                                                        -
!-------------------------------------------------------------------------

display_err     print           CLR_SCREEN
                select          error
                  case            1
                    print           E1_TXT
                    exit_select
                  case            2
                    print           E2_TXT
                    exit_select
                    default
                    print           NO_MATCH
                    exit_select
                    end_select
                let             time_1=200
                gosub           pause
                print           E3_TXT
```

```
de_ret_fkey    if_no_char     PORT,de_cont_fkey
de_clr_key     input          NULL,0,0,key,ENTER
de_wait_clr    if_flag_off    ENTER,de_wait_clr
               gosub          pause
               if_char        PORT,de_clr_key

de_cont_fkey   print          C_OFF

               input          NULL,0,0,key,ENTER
de_rfk_wait    if_flag_off    ENTER,de_rfk_wait
               if             key=F4,de_rfk_ok
               goto           de_ret_fkey

de_rfk_ok      print          C_ON
               return_sub

!-------------------------------------------------------------------------
!                                                                        -
!             Timer routine                                              -
!                                                                        -
!-------------------------------------------------------------------------
pause          set_tmr        TIMER_1,time_1
pausing        if_tmr_on      TIMER_1,pausing
               return_sub
```

Floating Point Numbers

```
!-------------------------------------------------------------------------
! TITLE:        Floating Point Numbers
!
! DATE:         12/3/97
!
! DESCRIPTION:  This program indexes AXIS_1 1" to 10" in 1" increments.
!               The roll diameter is 3.48" .
!
! EQUIPMENT:    DeltaMax
!
! COMMANDS:     float          float_dim
!
! The above commands are commonly used in many programs.  This
! illustration provides the context and syntax required to successfuly
! integrate these commands into a complete program.  While the use of
! these commands varies from program to program, the following example
! will give the programmer an indication of a typical usage.
!-------------------------------------------------------------------------

               declare        on

AXIS_1         equ            1
DOWN_1         equ            93
BUSY_1         equ            94


RESET          equ            0               ! resets system faults (input)
SYS_OK         equ            16              ! indicates system OK (output)


TIMER_1        equ            72
BITS_TURN      equ            4096

PI             equ            3.14159
ROLL_DIA       equ            3.48            ! Feed Roll diameter

cht            integer
cth            integer
feed           integer

feedbits       float
index_dis      float_dim      10              ! index distance array

start          let            cht=0
               let            cth=1

loop           let            feedbits=PI*ROLL_DIA
               let            feedbits=cth/feedbits
               let            feedbits=feedbits*BITS_TURN

               let            index_dis[cht]=feedbits
               let            cht=cht+1
               let            cth=cth+1
               if             cht<10,loop
```

```
                set_speed       AXIS_1,1000
                set_ac_dc       AXIS_1,200

                swi_if_on       0,DOWN_1,fault
                enable_swi
                drive_on        AXIS_1
                delta_comp      AXIS_1,1,1,30,0,1,6000

                let             cht=0
loop1           let             feed=index_dis[cht]

                index           AXIS_1,feed

wait1           if_stat_on      BUSY_1,wait1

                set_tmr         TIMER_1,50
wait2           if_tmr_on       TIMER_1,wait2

                let             cht=cht+1
                if              cht<10,loop1
                let             cht=0

                goto            loop1


!-------------------------------------------------------------------------

!                                                                       -
!  Subroutine used to handle a motor fault                              -
!                                                                       -

!-------------------------------------------------------------------------

fault           turn_off        SYS_OK
                f_decel         AXIS_1

                set_tmr         TIMER_1,200
wait_fault      if_tmr_on       TIMER_1,wait_fault

start_over      if_io_off       RESET,start_over
                drive_off       AXIS_1
                restart_at      start
```

MACROPRO II™

FOR WINDOWS™

Section 7 - Glossary

INDUSTRIAL INDEXING SYSTEMS, INC.

SECTION 7 - GLOSSARY

ACCEL/DECEL The rate of change to achieve a desired speed. Unit of measure is revs/sec^2.

ACTUATOR A device which creates mechanical motion by converting various forms of energy to mechanical energy.

AXIS An independent motor/drive unit which is controlled by an IIS type controller

AXIS CONTROLLER A controller which contained within the DeltaMax to control an independent motor/drive unit

CLOSED LOOP A regulating device in which the actuator position is sensed, and a signal proportional to this position (feedback position) is compared with a signal proportional to the desired actuator position (command position). The difference between these signals is the error signal. The error signal causes a change in the actuator so as to force this signal to be zero.

COMMAND SIGNAL GENERATOR A device that supplies a command position signal to an automatic control system. This signal represents the desired motion of the actuator that is required to accomplish a task such as making a part. This signal is usually in the form of an electrical signal.

COMMUNICATIONS The transmission of information from one device to another. The information can take many forms such as command signals, device status and fault conditions.

COMMUTATOR The part of the rotating armature of a motor that causes the electrical current to be switched to various armature windings. The proper sequenced switching of the windings creates the motor torque. The commutator also provides the means to transmit the electrical current from the stationary body of the motor to moving rotor.

COMPARATOR A device where the feedback signal is subtracted from the command signal. The difference output of the comparator is called the error signal.

DC DRIVE An electronic control unit for running DC motors. The DC drive converts AC line current to a variable DC current to control a DC motor. The DC drive has a signal input that controls the torque and speed of the motor.

ENCODER A type of feedback device which converts mechanical motion into electrical signals to indicate actuator position. Typical encoders are designed with a printed disc and a light source. As the disc turns with the actuator shaft, the light source shines through the printed pattern onto a sensor. The light transmission is interrupted by the patterns on the disk. These interruptions are sensed and converted to electrical pulses. By counting these pulses, actuator shaft position is determined.

FLAG A bit in memory used by the programmer to evaluate action to be taken. A program branch may be executed depending on the true or false result of a bit test. There are four types of flags supported by the IIS programming language; axis status flags, I/O status flags, timer status flags and user flags.

HOME A known point of reference frequently referred to as "global zero". Used as a point of reference for absolute positioning.

HOST A computer system whose function is to monitor and coordinate the processes of other devices. A host computer will typically coordinate motion control functions as well as their interaction with other machine processes.

I/O The reception and transmission of information (Input/Output) between control devices. In modern control systems, I/O has two distinct forms. In one, it refers to switches, relays, etc. In the other form, I/O refers to analog signals that are continuous in nature such as speed, temperature, flow, etc.

INDEX An **index** instruction will move the motor shaft an absolute distance relative to the current position.

INERTIA The measure of an object's resistance to a change in its current velocity.

INITIALIZE To execute a series of program instructions in order to teach the DeltaMax motor axis controller an absolute zero reference.

INSTRUCTION A program command to the DeltaMax. All instructions have the following format:

label instruction parameters comment

INSTABILITY Undesirable motion of an actuator that is different from the commanded motion. Instability can take the form of irregular speed or hunting of the final rest position.

JOG

A **jog** instruction will move the motor shaft in the specified direction using the predefined speed and accel/decel rates. Once at speed, the motor will continue to jog until commanded to stop.

LINE SHAFT

A shaft rotated by the primary motor drive. The line shaft transmits power from the motor to a load or series of loads. In the multiple load case, the motions of the loads are synchronized to one another because they are driven from a common shaft.

MACROPRO II

The Windows-based software package used to edit, compile, and analyze programs developed for DeltaMax controllers.

MICROPROCESSOR

A miniaturized computer system that executes instructions in a sequential manner. The sequential instructions form a control strategy for devices that may be connected to the system.

The sequential instructions are loaded into the microprocessor and can be easily changed or modified. Modern microprocessors are small electronic devices that execute a wide range of instructions at speeds as high as 1,000,000 instructions per second.

MOTOR DIRECTION

The direction that a motor shaft is turning is determined by the facing the front of the motor shaft. From this perspective, direction is determined to be clockwise or counter-clockwise.

OPEN-LOOP SYSTEM

A system where a command signal results in actuator movement which is not sensed and therefore not corrected for error. The term "open loop" essentially means there is no feedback.

OPERATOR INTERFACE

A device that allows the operator to communicate with a machine. This device typically has a keyboard or thumbwheel to enter instructions into the machine. It may also have a display device that allows the machine to display messages. An example of an operator interface is the OPI-1.

OSCILLATION

Undesirable motion of an actuator that is different from the command motion. See INSTABILITY.

PARAMETER A value required for correct execution of an instruction. Each instruction has a parameter list which is used by the controller to execute the instruction. An example would be the **index** instruction. The controller needs to know which axis to index and the distance to index.

PERIPHERAL Various kinds of devices that operate in conjunction with a DeltaMax controller. Examples of these are PLC's, joysticks, computers, and operator interfaces.

POSITION ERROR The difference between the present actuator position (feedback) and the desired position (command).

POSITION FEEDBACK Present actuator position as measured by a position transducer.

PROGRAM / MACROPROGRAM A program written in IIS's basic-like language for the DeltaMax and MSC-family of motion controllers.

PROGRAMMABLE LOGIC CONTROLLER (PLC) (PLC) An electronic device that scans discrete (on/off) type inputs and controls discrete (on/off) type outputs. The relationship between the inputs and outputs are logical statements that are programmable by the user.

RESOLVER A type of feedback device which converts mechanical position into an electrical signal. A resolver is a variable transformer that divides the impressed AC signal into two outputs, referred to as sine and cosine output signals. The comparison of these two signals is used to determine the absolute position of the resolver shaft.

ROTOR The rotating part of a magnetic structure. In a motor, the rotor is connected to the motor shaft.

SERVO An automatic control device for controlling large amounts of power by means of very small amounts of power and automatically correcting the performance of the mechanism. Servo systems are closed loop systems.

SERVO AMPLIFIER An electronic device which produces the winding current for a servo motor. The amplifier converts a low level control signal into high voltage and current levels to produce torque in the motor.

SERVO MOTOR

An actuator which converts electrical energy (winding current) to mechanical energy (torque). Servo motor construction is optimized to provide maximum torque with minimum rotor inertia.

MACROPRO II™

FOR WINDOWS™

Section 8 - Appendix

INDUSTRIAL INDEXING SYSTEMS, INC.

SECTION 8 - APPENDIX

Table of Contents

This page is intentionally left blank.

Adjustment Parameters

ADJUSTMENT PARAMETER	SYMBOL	DRIVE RANGE	DELTAMAX DEFAULT	DESCRIPTION
LOAD INERTIA RATIO	AJ2	0-100.0 TIMES	1.0 TIMES	Sets the baseline frequency response of the driver using the ratio of the load inertia/motor inertia for a rigidly coupled load. If the load is not rigidly coupled, the value entered may vary from the calculated value. If the value is set too high, the motor and driver may become unstable and oscillate.
HIGH FREQUENCY RESPONSE	AJ3	0.1-20.0	1.0	Sets the high frequency response of the driver. The higher the number the more responsive. If the value is set too high, the motor and driver may become unstable and oscillate. The value in AJ3 is unitless and works in concert with AJ2.
POSITION LOOP DC GAIN	AJ4	1-200 RAD/SEC	30 RAD/SEC	Sets the DC gain of the position control loop. A higher value in AJ4 results in stiffer, faster response. If the value is set too high, the motor and driver may become unstable & oscillate.
ZERO SPEED GAIN REDUCTION	AJ7	0-10000	0	Sets the amount of gain reduction at zero speed.
FEED FORWARD GAIN	AJ8	0-2.0 TIMES	1.0 TIMES	Sets the feed forward gain in the position loop. A value of 1.0 results in 0.0 following error. Less than 1.0 will produce a lag between the actual motor position and the commanded position and greater than 1.0 produces a lead. The lead or lag will be proportional to speed at non 1.0 settings.
NOTCH FILTER FREQUENCY	AJ9	100-20000 RAD/SEC	6000 RAD/SEC	Sets the notch frequency of a velocity loop anti-resonance filter. This filter can be used to cancel machine or servo resonance. Power must be turned OFF then ON for this parameter to take effect.

Electronic Data Sheet (EDS)

The Electronic Data Sheet (EDS) allows a configuration tool to automate the DeviceNet device configuration process. An EDS contains vendor specific and/or MacroProgram specific information defined for a particular application. The EDS file may be used to easily get a DeltaMax up on the DeviceNet Network.

An EDS definition is available by contacting ODVA, or obtaining Section 2 Chapter 4 of the ODVA DeviceNet Specification.

The following EDS file example was generated using DeviceNet Manager and a simple text editor. DeviceNet Manager is a DeviceNet software tool distributed by Allen Bradley. The example is intended to work with the MacroPragram exampled without a parameter object, herein.

If a parameter object is macroprogrammed into the DeltaMax then a EDS file maybe generated automatically from a DeviceNet such as DeviceNet Manager. The tool exstracts the necessary information from the DeltaMax, then writes the file to the PC.

EDS File Example

```
$ DeviceNet Manager Generated Electronic Data Sheet
$

[File]


[Device]
      VendCode    = 89;                                   $ Vendor Code
      ProdType    = 0;                                    $ Product Type
      ProdCode    = 1;                                    $ Product Code

      MajRev      = 1;                                    $ Major Rev
      MinRev      = 0;                                    $ Minor Rev
      VendName    = "Industrial Indexing Systems, Inc";
      ProdTypeStr = "Generic";
      ProdName    = "DeltaMax";
      Catalog     = "0";

[IO_Info]
      Default     = 0X0000;


[ParamClass]
      MaxInst=4;
      Descriptor=0;
      CfgAssembly=0;

[Params]

   $ Flags Object Parameters
        Param1=                                 $ class:0x64,Inst:0x21,Attr:0x3
                0,                              $ parameter value slot
                6, "20 64 24 21 30 03",         $ link path size and path
                0x0002,                         $ descriptor: enumerated
                4,                              $ data type: BOOLEAN
                1,                              $ data size: 1 for BOOLEAN
                "RUN INPUT",                    $ parameter name
                "",                             $ units string
                "",                             $ help string
```

```
                0, 1, 0,                        $ min, max, default
                1, 1, 1, 0,                     $ mult, div, base, offset scaling
                0, 0, 0, 0, 0 ;                 $ scaling links and precision

   $ Integer Object Dynamic Parameters
        Param2=                                 $ class:0x65,Inst:0x01,Attr:0x3
                0,                              $ parameter value slot
                6, "20 65 24 01 30 03",         $ link path size and path
                0x0000,                         $ descriptor: none
                6,                              $ data type: DINT
                4,                              $ data size: 4 for DINT
                "Speed",                        $ parameter name
                "rpm",                          $ units string
                "",                             $ help string
                0, 3000, 100,                   $ min, max, default
                1, 1, 1, 0,                     $ mult, div, base, offset scaling
                0, 0, 0, 0, 0 ;                 $ scaling links and precision

        Param3=                                 $ class:0x65,Inst:0x03,Attr:0x3
                0,                              $ parameter value slot
                6, "20 65 24 03 30 03",         $ link path size and path
                0x0000,                         $ descriptor: none
                6,                              $ data type: DINT
                4,                              $ data size: 4 for DINT
                "Multipler",                    $ parameter name
                "",                             $ units string
                "",                             $ help string
                0, 4, 1,                        $ min, max, default
                1, 1, 1, 0,                     $ mult, div, base, offset scaling
                0, 0, 0, 0, 0 ;                 $ scaling links and precision

        Param4=                                 $ class:0x65,Inst:0x0d,Attr:0x3
                0,                              $ parameter value slot
                6, "20 65 24 0d 30 03",         $ link path size and path
                0x0010,                         $ descriptor: readonly
                6,                              $ data type: DINT
                4,                              $ data size: 4 for DINT
                "Angle * Multiplier",           $ parameter name
                "bits(4096bits/rev)",           $ units string
                "",                             $ help string
                0, 4095, 0,                     $ min, max, default
                1, 1, 1, 0,                     $ mult, div, base, offset scaling
                0, 0, 0, 0, 0 ;                 $ scaling links and precision

[EnumPar]

   $ Flags Object Parameters Enumerations
        Param1=                                 $ Bit Definitions for Param 1
                "OFF",
                "ON";


[Groups]
```

DeltaMax Device Data

General Device Data

Device Data					
General Device Data	Conforms to Devicenet Specification	Volume I – Release 1.3			
		Volume II – Release	1.2		
	Vendor Name	Industrial Indexing Systems, Inc.			
	Device Profile Name	Generic Device			
	Product Name	DeltaMax			
	Product Catalog Number	0			
	Product Revision	1			
DeviceNet Physical Conformance Data	Network Power Consumption (Max)	0.12 A@11V dc (worst case)			
	Connector Style	Open–Hardwired	☐	Sealed-Mini	☐
		Open–Pluggable	☑	Sealed-Micro	☐
	Isolated Physical Layer	Yes	☑		
		No	☐		
	LEDs Supported	Module	☐	Combo Mod/Net	☑
		Network	☐	I/O	☐
	MAC ID Setting	DIP Switch	☐	Software Settable	☑
		Other MacroProgrammable			
	Default MAC ID	63			
	Communication Rate Setting	DIP Switch	☐	Software Settable	☑
		Other MacroProgrammable			
	Communication Rates Supported	125k bit/s	☑	500k bit/s	☑
		250k bit/s	☑		
DeviceNet Communication Data	Device Network Behavior Check All That Apply	Group 2 Client	☐	Group 2 Only Client	☐
		Group 2 Server	☐	Group 2 Only Server	☑
		Peer-to-Peer	☐	Tool (not a device)	☐
	UCMM Explicit Message Groups Supported	Group 1 ☐	Group 2 ☐	Group 3 ☐	
	Dynamic I/O Message Groups	Group 1 ☐	Group 2 ☐	Group 3 ☐	
	Typical I/O Data Address Path	Input: Class 4	Inst. 1	Attr. 3	
		Output: Class 4	Inst. 2	Attr. 3	
	Fragmented Explicit Messaging Implemented	Yes	☑		
	If yes, Acknowledge Time Out	No	☐		
	Typical Target Addresses (decimal)				
	Consumption	Service 16	Class 1	Inst. 1	Attr. 7
	Production	Service 16	Class 1	Inst. 1	Attr. 7

Identity Object

DeviceNet	Identity Object 0x01						
Required	**Object Class**		**ID**	**Description**	**Get**	**Set**	**Value Limits**
Object	Attributes	Open	1	Revision	☑	☐	
Implementation			2	Max instance	☐	☐	
	☐ None Supported		6	Max ID of class attributes	☐	☐	
			7	Max ID of instance attributes	☐	☐	
			DeviceNet Services		**Parameter Options**		
	Services		☐	Get_Attribute_All			
			☐	Reset			
	☐ None Supported		☑	Get_Attribute_Single			
			☐	Find_Next_Object_Instance			
	Object Instance		**ID**	**Description**	**Get**	**Set**	**Value Limits**
	Attributes	Open	1	Vendor	☑	☐	89
			2	Product type	☑	☐	0
	☐ None Supported		3	Product code	☑	☐	0
			4	Revision	☑	☐	1
			5	Status	☑	☐	
			6	Serial number	☑	☐	
			7	Product name	☑	☐	DeltaMax
			8	State	☐	☐	
				DeviceNet Services	**Parameter Options**		
	Services		☐	Get_Attribute_All			
	☐ None Supported		☑	Reset	0		
			☑	Get_Attribute_Single			
	Vendor Specific Additions		If yes, fill out the Vendor Specific Additions form on page F– 7.		Yes ☐		
					No ☑		

Message Router Object

DeviceNet Required Object Implementation	Message Router Object 0x02						
	Object Class		**ID**	**Description**	**Get**	**Set**	**Value Limits**
	Attributes	Open	1	Revision	☐	☐	___
			4	Optional attribute list	☐	☐	___
	☒None Supported		5	Optional service list	☐	☐	___
			6	Max ID of class attributes	☐	☐	___
			7	Max ID of instance attributes	☐	☐	___
			DeviceNet Services		**Parameter Options**		
	Services		☐	Get_Attribute_All	___		
			☐	Get_Attribute_Single	___		
	☒None Supported						
	Object Instance		**ID**	**Description**	**Get**	**Set**	**Value Limits**
	Attributes	Open	1	Object list	☐	☐	___
			2	Maximum connections supported	☐	☐	___
	☒None Supported		3	Number of active connections	☐	☐	___
			4	Active connections list	☐	☐	___
			DeviceNet Services		**Parameter Options**		
	Services		☐	Get_Attribute_All	___		
	☒None Supported		☐	Get_Attribute_Single	___		
	Vendor Specific Additions		If yes, fill out the Vendor Specific Additions form on page F– 7.		Yes	☐	
					No	☒	

DeviceNet Object

DeviceNet	DeviceNet Object 0x03						
Required	**Object Class**		**ID**	**Description**	**Get**	**Set**	**Value Limits**
Object	Attributes	Open	1	Revision	☐	☐	___
Implementation	☒None Supported						
			DeviceNet Services		**Parameter Options**		
	Services		☐	Get_Attribute_Single	___		
	☒None Supported						
	Object Instance		**ID**	**Description**	**Get**	**Set**	**Value Limits**
	Attributes	Open	1	MAC ID	☒	☒	0 to 63
			2	Baud rate	☒	☒	0 to 2
	☐ None Supported		3	BOI	☒	☐	___
			4	Bus–off counter	☒	☒	0
			5	Allocation information	☒	☐	___
			6	MAC ID switch changed	☒	☐	___
			7	Baud rate switch changed	☐	☐	___
			8	MAC ID switch value	☐	☐	___
			9	Baud rate switch value	☐	☐	___
			DeviceNet Services		**Parameter Options**		
	Services		☒	Get_Attribute_Single	___		
			☒	Set_Attribute_Single	___		
	☐ None Supported		☒	Allocate M/S connection set	___		
			☒	Release M/S connection set	___		
	Vendor Specific Additions		If yes, fill out the Vendor Specific Additions form on page F– 7.		Yes	☐	
					No	☒	

Connection Object - Explicit Message

DeviceNet Required Object Implementation

Connection Object 0x05

Object Class		ID	Description	Get	Set	Value Limits
Attributes	Open	1	Revision	☐	☐	__
None Supported						

	DeviceNet Services	Parameter Options
Services	☐ Reset	___
	☐ Create	___
None Supported	☐ Delete	___
	☐ Get_Attribute_Single	___
	☐ Find_Next_Object_Instance	___

Total Active Connections Possible **3**

Object Instance 1	Section	Information	Max Instance
The Object Instance section must be completed for each combination of Instance type, Production trigger, Transport type, and Transport class supported	Instance type	Explicit Message	1
		Polled I/O ☐	
		Bit Strobed I/O ☐	
		Dynamic I/O ☐	
	Production trigger	Cyclic ☐	
		Change of State ☐	
		Application Triggered ☐	
	Transport type	Server ☐	
		Client ☐	
	Transport class	0 ☐	
		2 ☐	
		3 ☐	

Attributes	Open	ID	Description	Get	Set	Value Limits
		1	State		☐	__
		2	Instance type		☐	__
		3	Transport class trigger		☐	__
		4	Produced connection ID		☐	__
		5	Consumed connection ID		☐	__
		6	Initial comm. characteristics		☐	__
		7	Produced connection size		☐	__
		8	Consumed connection size		☐	__
		9	Expected packet rate			__
		12	Watchdog time–out action		☐	__
		13	Produced connection path length		☐	__
		14	Produced connection path		☐	__
		15	Consumed connection path length		☐	__
		16	Consumed connection path		☐	__

	DeviceNet Services	Parameter Options
Services	☒ Reset	___
	☐ Delete	___
	☐ Apply_Attributes	___
	☒ Get_Attribute_Single	___
	☒ Set_Attribute_Single	___
Vendor Specific Additions	If yes, fill out the Vendor Specific Additions form on page F– 7.	Yes ☐ No ☒

Connection Object - Polled I/O

Connection Object 0x05 (Continued)

Object Instance 2	Section	Information		Max Instance
The Object Instance section must be completed for each combination of Instance type, Production trigger, Transport type, and Transport class supported	Instance type	Explicit Message	☐	1
		Polled I/O	☑	
		Bit Strobed I/O	☐	
		Dynamic I/O	☐	
	Production trigger	Cyclic	☐	
		Change of State	☐	
		Application Triggered	☐	
	Transport type	Server	☐	
		Client	☐	
	Transport class	0	☐	
		2	☐	
		3	☐	

		ID	Description	Get	Set	Value Limits
Attributes	Open	1	State	☑	☐	
		2	Instance type	☑	☐	
		3	Transport class trigger	☑	☐	
		4	Produced connection ID	☑	☐	
		5	Consumed connection ID	☑	☐	
		6	Initial comm. characteristics	☑	☐	
		7	Produced connection size	☑	☐	
		8	Consumed connection size	☑	☐	
		9	Expected packet rate	☑	☑	
		12	Watchdog time–out action	☑	☐	
		13	Produced connection path length	☑	☐	
		14	Produced connection path	☑	☐	
		15	Consumed connection path length	☑	☐	
		16	Consumed connection path	☑	☐	

	DeviceNet Services	Parameter Options
Services	☑ Reset	
	☐ Delete	
	☐ Apply_Attributes	
	☑ Get_Attribute_Single	
	☑ Set_Attribute_Single	

Vendor Specific Additions	If yes, fill out the Vendor Specific Additions form on page F– 7.	Yes	☐
		No	☑

Connection Object - Bit Strobed I/O

Connection Object 0x05 (Continued)

Object Instance 3	**Section**	**Information**	**Max Instance**
The Object Instance section must be completed for each combination of Instance type, Production trigger, Transport type, and Transport class supported	Instance type	Explicit Message ☐ Polled I/O ☐ Bit Strobed I/O ☑ Dynamic I/O ☐	1
	Production trigger	Cyclic ☐ Change of State ☐ Application Triggered ☐	
	Transport type	Server ☐ Client ☐	
	Transport class	0 ☐ 2 ☐ 3 ☐	

		ID	**Description**	**Get**	**Set**	**Value Limits**
Attributes	Open	1	State	☑	☐	___
		2	Instance type	☑	☐	___
		3	Transport class trigger	☑	☐	___
		4	Produced connection ID	☑	☐	___
		5	Consumed connection ID	☑	☐	___
		6	Initial comm. characteristics	☑	☐	___
		7	Produced connection size	☑	☐	___
		8	Consumed connection size	☑	☐	___
		9	Expected packet rate	☑	☑	___
		12	Watchdog time–out action	☑	☐	___
		13	Produced connection path length	☑	☐	___
		14	Produced connection path	☑	☐	___
		15	Consumed connection path length	☑	☐	___
		16	Consumed connection path	☑	☐	___

	DeviceNet Services	**Parameter Options**
Services	☑ Reset	___
	☐ Delete	___
	☐ Apply_Attributes	___
	☑ Get_Attribute_Single	___
	☑ Set_Attribute_Single	___

Vendor Specific Additions	If yes, fill out the Vendor Specific Additions form on page F– 7.	Yes ☐ No ☑

Assembly Object

DeviceNet	**OBJECT NAME** Assembly Object				**OBJECT ID** 0x04	
Open	**Object Class**		**ID**	**Description**	**Get** **Set**	**Value Limits**
Object	Attributes	Open	1	Revision	☐ ☐	1
Implementation	☒ None Supported		2	Max instance	☐ ☐	
			DeviceNet Services		**Parameter Options**	
	Services		☐	Create		
	☒ None Supported		☐	Delete		
			☒	Get_Attribute_Single		
	Object Instance		**Section**		**Information**	**Max Instance**
	The Object Instance section must be completed for each combination of Instance type.		Instance type		Static Input	☐
					Static Output	☐
					Static I/O	☒
	☐ None Supported				Static Configuration	☐
			ID	**Description**	**Get** **Set**	**Value Limits**
	Attributes	Open	1	Number of members in list	☐ ☐	
	☐ None Supported		2	Member list	☐ ☐	
			3	Data	☒ ☒	
			DeviceNet Services		**Parameter Options**	
	Services		☐	Delete		
	☐ None Supported		☒	Get_Attribute_Single		
			☒	Set_Attribute_Single		
			☐	Add_Member		
			☐	Delete_Member		
	Vendor Specific Additions		If yes, fill out the Vendor Specific Additions form on page F– 7.		Yes ☐ No ☒	