DSPL 5.0

A Guide to Programming Mx4 in DSPL Language

Turbo DSPL

Programmer's Guide

v5.0

DSP Control Group, Inc.
PO Box 39331
Minneapolis, MN 55435
Phone: (952) 831-9556
FAX: (952) 831-4697

The authors and those involved in the manual's production have made every effort to provide accurate, useful information.

Use of this product in an electro mechanical system could result in a mechanical motion that could cause harm. DSP Control Group, Inc. is not responsible for any accident resulting from misuse of its products.

Contents

This page intentionally blank.

1 Introduction

Congratulations on purchasing a DSP Control Group's high-speed multi-DSP motion controller. You will find DSPL a powerful language with an instruction set suitable for all coordinated motion control applications.

This manual contains additional information specific to Turbo DSPL, a version of DSPL which maximizes instruction throughput for higher performance. Instruction listings in Chapter 8 include instruction timing information.

The DSPL Programmer’s Guide supports the 2-axis Mx42, the 4-axis Mx4, and the 8-axis Mx4 Octavia controllers. Unless otherwise noted, descriptions are provided for the 8-axis Mx4 Octavia. When this manual is used in conjunction with the 4-axis Mx4 or the 2-axis Mx42, remember that the axes available are 1-4 for the Mx4 and 1-2 for the Mx42 (rather than 1-8 for the Mx4 Octavia).

Also note that througout this manual, unless otherwise noted, the term Mx4 will be used to refer generically to all three controllers.

In addition to this manual, you may find the following manuals helpful:

Mx42 User’s Guide

Mx4 User's Guide

Mx4 Octavia User's Guide

These manuals include comprehensive information on Mx42/Mx4/Mx4 Octavia's hardware, software, system tuning, memory organization, trouble shooting, and more. The *User's Guide* is the focal point in learning the technical details of these products. All other manuals assume that the user has familiarity with these manuals.

Mx4Pro Development Tools

This manual describes Mx4Pro - a testing and tuning software program used with Mx42, Mx4, and Mx4 Octavia. Mx4Pro includes features such as a signal generator, oscilloscope, and live block diagram which make the program useful for testing and performance optimization.

Vx4++ User's Guide

This manual includes information on the add-on drive control option. Vx4++ is DSPCG's multi-DSP based drive controller that provides complete drive signal processing for all industrial DC and AC machines. Vx4++ has capabilities that are normally offered by servo control amplifiers.

Mx4 & Windows

If your motion application operates under the Windows 95 or Windows NT operating system, you will want to utilize the Mx4 DLL. The *Mx4 & Windows* manual accompanies the DLL, providing information for both Visual Basic and C/C++ programming. The Mx4 DLL includes functionality in all aspects of Mx42 / Mx4 / Mx4 Octavia use, including utilities for DSPL downloading, DSPL execution start and stop, and much more.

2 Installation

The Mx4Pro Development Tools include DSPL Program Development as an integrated part of the Tools. The Mx4Pro Development Tools provide both first-time and experienced DSPL programmers with easy access to a host of powerful development aids, ranging from simple DSPL tutorials to compensation table download utilities for more advanced applications. As such, it is strongly recommended that the Mx4Pro Development Tools be used for DSPL program development. Within Mx4Pro, the DSPL Program Development environment may be invoked via the DSPL icon on the main Mx4pro Development Tools tool bar. Please refer to the *Mx4Pro Development Tools v5.x* manual for software installation details.

Chapter 6, *DSPL Program Development*, contains helpful information which details the use of the DSPL Program Development environment within the Mx4Pro Development Tools.

DSPL program development may also be integrated into any Windows 95 or Windows NT application via the DSPL utilities provided in the Mx4 DLL. Refer to the *Mx4 & Windows* manual or contact DSPCG for more information.

This page intentionally blank.

3 Methods of Programming Mx4

Before we immerse ourselves in the specifics of DSPL programming. let's look at the two different methods of programming the Mx4 controller. DSP Control Group has applied years of experience in the motion control industry to the development of Mx4's dual programming platform. Mx4 may be programmed via real time Host-based programming, or at a DSPL (internal language of Mx4) level, or a combination of both.

Host-Based Programming

Host-based programming entails real-time communication between the host computer and the Mx4 card across the host computer bus. This communication originates from an Mx4 motion application running on the host computer. The host computer may read and write to the Mx4 card as it would any computer peripheral. The user chooses the programming language for the host computer program. For example, it may be a DOS application written in C, or maybe a Visual Basic Windows NT application. DSPCG provides programming utilities ranging from C functions to Visual Basic / C DLLs for host-based program development. This host program includes the following: facilities to transfer commands to the Mx4 card through the host bus, any conditional program code execution routines, PLC emulation code, an optional interrupt service routine to handle any enabled Mx4 interrupts, Mx4 system parameter readback routines, plus any other software features required for the application. When using host programming, an executable host program runs the operation of the Mx4 card in real time.

Note: Mx4 Host programming is described in detail in the *Mx4 User's Guide*. This document, the *DSPL Programmer's Guide*, focuses on Mx4 DSPL programming.

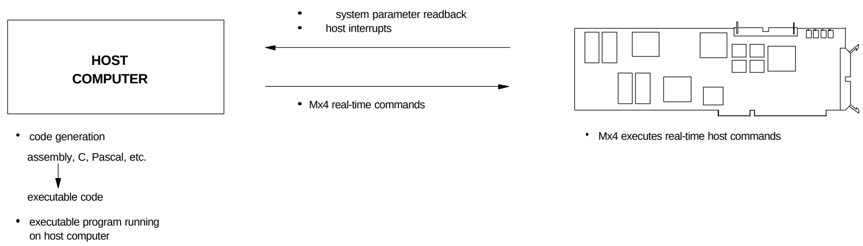


Fig. 3-1: Mx4 Host-Based Programming

DSPL Programming

The Mx4's high-level DSPL programming platform enables complete motion control applications to be written in the DSPL programming language, downloaded once to the Mx4 card, and executed by the Mx4 card. The DSPL programming language is a powerful, full-featured, yet easy to use language that includes features such as conditional program execution, subroutine calls, separate PLC and motion programming facilities, and the ability to run PLC and multiple Motion programs simultaneously on the Mx4 card.

A DSPL program consists of a text file which may be written with any text editor. The DSPL code is then compiled and downloaded to Mx4's memory. With the use of the optional non-volatile battery-backup memory available for Mx4, standalone operation is possible once the DSPL program is downloaded to the card. Once the DSPL code is loaded into Mx4's memory, Mx4 may begin executing the code. DSPL code execution by Mx4 is independent of the host computer.

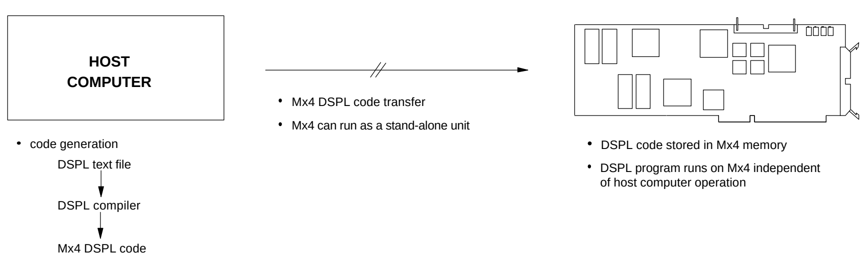


Fig. 3-2: Mx4 DSPL Programming

Combining DSPL & Host-Based Programming

Although both the Host and DSPL Mx4 programming techniques are full featured and self-supporting, you may choose to combine the two, drawing the advantages of both techniques in solving a particular programming application. While running or executing DSPL PLC and Motion programs, Mx4 is still completely programmable via the host (Host-based programming methods). This feature of Mx4 allows for a combination of Host and DSPL programming. In addition, a synchronizing timing structure may be established between an executing DSPL program and the host computer via Mx4's powerful command sets.

Introduction to DSPL Programming

DSPL was designed to combine the flexibility of low-level instructions with the convenience of a high-level language. To use DSPL, only a minimum programming background is required, since DSPL only contains common sense language constructs. If you are a first time DSPL programmer, you will find yourself writing simple applications in minutes with the aid of the Mx4pro Development Tools and included tutorials.

DSPL is a powerful programming language designed to take advantage of Mx4's multi-DSP architecture and multi-tasking capabilities. DSPL includes low and high-level instructions that make it ideal for both simple and more advanced motion control programming.

A typical DSPL program consists of two distinct portions, PLC programming code and Motion programming code. A DSPL program always includes a single PLC sub-program and any number of Motion sub-programs (Fig. 3-3). (In this manual the PLC and Motion sub-programs will be referred to as PLC and Motion programs).

```
plc_program:
        initialize parameters and gains
        perform logical operations on I/Os
        start motion programs
end
        motion_prog_1:
                do simple and coordinated moves
                perform logical operations
                do conditional branching
                call subroutines
                check interrupts
        end
        motion_prog_2:

        end
        .
        .
        .
        motion_prog_n:

        end
```

Fig. 3-3: A Typical DSPL Program Sheet

Mx4 is capable of running the PLC program and up to two Motion programs simultaneously. Mx4 Octavia is capable of running the PLC program and up to three Motion programs simultaneously.

PLC Programs

The PLC program is typically used as a "monitor" program emulating a Programmable Logic Controller. As is indicated in Fig. 3-3, the PLC may be used to execute initialization routines, monitor system status, perform logical operations based on input/output, run Motion programs, perform conditional Motion program execution, and many more application-specific functions.

Based on a logical combination of inputs and/or dynamic system state values (e.g., position, position error, or velocity), the PLC can make an executive decision. The decision can be as simple as setting an output bit or executing one or several motion programs simultaneously.

As an example, consider the following simple PLC program.

```
PLC_PROGRAM

        #include "INIT.hll"

        VAR1 = 0
        run_m_program (INIT_MX4)
        wait_until (INP1_REG & 0x0001)
        run_m_program (PROFILE_1)

END
```

This PLC program, although very simple, illustrates some important fundamentals of PLC programming such as variable and system initialization and conditional Motion program execution.

Motion Programs

The Mx4's multi-tasking operating system allows simultaneous execution of the PLC program and up to three Motion programs. DSPL Motion programs consist of either conditional or unconditional execution of DSPL commands (both motion and non-motion related), logical operations, conditional branching, subroutine calls, the issuance of interrupts, etc. A Motion program is initiated by the PLC program, but runs independent of the PLC.

Motion programs may contain I/O instructions similar to those found traditionally in the PLC. The Motion programs resemble C code and include common logical and conditional constructs such as if, endif, while, wend, etc. A Motion program can include several hundred lines of high-level commands, or, in a shorter form, can include several calls to subroutines performing a dedicated task.

The following is an example of a simple Motion program.

```
SEG_A1_TO_B1:

       pos_preset (0x3,1000,3500)

       if ((POS3 > 500) and (CVEL1 = 0))
              linearmove (0x2,0,0,1000,1.0,2,0.025)
              circle (0xC,0,1000,500,0.75,0,0)
       endif

END
```

4 Mx4 DSPL Programming

As we have seen, a DSPL program consists of two parts: the PLC sub-program and the Motion sub-program(s) (Fig. 4-1)

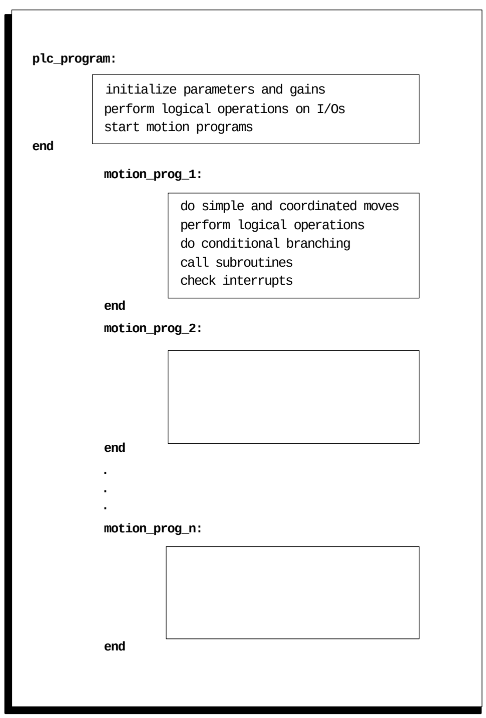


Fig. 4-1: A Typical DSPL Program Sheet

The PLC and Motion programs together are collectively referred to as a DSPL program. The DSPL program is merely a text file, which is then compiled and downloaded to the Mx4 card. The following sections illustrate some of the basics of DSPL programming.

DSPL Programming Basics

Program Entry

The DSPL program is a text file containing a series of DSPL commands, keywords, and operators, which make up the PLC and (any number of) Motion programs. A DSPL program may consist of a maximum of 2048 DSPL command lines. The DSPL program may be entered with any standard text editor via the Mx4pro Development Tool (see Chapter 6, *DSPL Program Development*).

The DSPL program file must be a suffix of .hll. For example:

filename.hll

Note: The .hll suffix is required in order for the DSPL program file to be compiled by the DSPL compiler.

Syntax

Note: The syntax for the usage of individual DSPL commands is included in the listing of each of the commands (see *DSPL Command Set*).

The DSPL programming language follows some very simple structural syntax rules.

Upper & Lower Case Characters

DSPL programs may be written in either upper or lower case characters, or any combination of such. The DSPL compiler does not differentiate between upper and lower case. The following example Motion program illustrates this point,

```
EXAMPLE:

        var1=1
        VAR2=33

        if(inp1_REG&0x0010)
                maxacc(0x1,0.024)
                VELMODE(0x1,6.5)
        ENDIF

end
```

In order to ease program readability, it is advisable that the programmer follows a procedure for the use of upper and lower case characters. For example, the programmer may wish to reserve upper case characters for program labels and variable designators,

```
EXAMPLE:

        VAR1=1
        VAR2=33

        if(INP1_REG&0x0010)
                maxacc(0x1,0.024)
                velmode(0x1,6.5)
        endif

END
```

Blank Space

The DSPL compiler does not require any spacing or carriage returns between commands. For example, the following example Motion program is a valid program,

```
EXAMPLE:       VAR1=1VAR2=33         if(INP1_REG&   0x0010)
maxacc                 (0x1,   0.024)velmode
        (0x1,6.5)
endif END
```

Again, it is strongly advised that the programmer use a spacing procedure with spaces, tabs, and/or carriage returns in order to increase readability of the program as well as to indicate program flow and structure.

```
EXAMPLE:

       VAR1 = 1
       VAR2 = 33

       if (INP1_REG & 0x0010)
              maxacc (0x1,0.024)
              velmode (0x1,6.5)
       endif

END
```

Commenting Programs

It is often convenient to place comments or notes in a program in order to improve the program's readability. In DSPL a comment always begins with a semi-colon (;) and ends with a carriage return. For example,

```
;This program is an example

EXAMPLE:

       VAR1 = 1       ;initialize variable 1
       VAR2 = 33      ;define VAR2=33

       if (INP1_REG & 0x0010)        ;if IN1(1) input is
              maxacc (0x1,0.024)     ;set, then initiate
              velmode (0x1,6.5)      ;velocity mode motion
       endif

END
```

Writing PLC Programs

What is a PLC Program?

Each DSPL program must include a single PLC program. The PLC (or Programmable Logic Controller) program is typically used as a monitor program, utilizing input logic and/or system parameter conditions for evaluating conditional expressions, and initiating the execution of Motion programs.

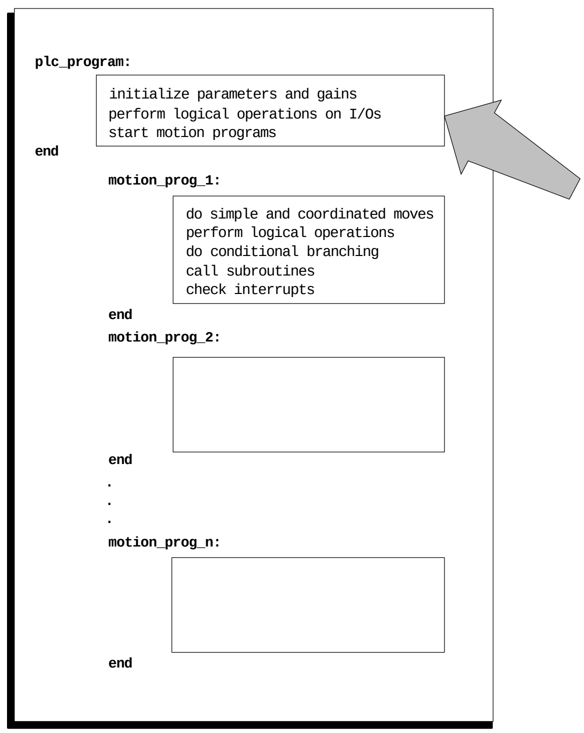
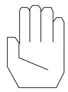


Fig. 4-2: A Typical DSPL Program Sheet, PLC Program Highlighted

Due to its “monitoring” function, the PLC program must execute in an uninhibited fashion. For this reason, the PLC program is limited as to the DSPL commands, which may appear within it. For example, a DELAY command is not allowed in the PLC program, since the PLC program code execution halts during the specified duration of the DELAY command, impairing the PLC "monitoring" function. Also, motion and system commands are restricted from use in the PLC program. In short, only those commands, operators, and keywords related to system initialization, conditional expression evaluation, and Motion program execution are available to the PLC program.

The DSPL command listings (see *DSPL Command Set*) include a USAGE category that indicates whether or not the command is available for use in the PLC program. The following table indicates the PLC and/or Motion program usage of the DSPL commands.



Note : Operators and identifiers are not PLC/Motion program sensitive.

DSPL COMMANDS	PLC	MOTION
ABS	✓	✓
ADC1, ADC2, ADC3, ADC4	✓	✓
AND, OR	✓	✓
ARCTAN	✓	✓
AXMOVE		✓
AXMOVE_S		✓
AXMOVE_T		✓
BTRATE		✓
CALL		✓
CAM		✓
CAM_OFF		✓
CAM_OFF_ACC		✓
CAM_POINT		✓
CAM_POS		✓
CAM_PROBE		✓
CAMCOUNT1, …, CAMCOUNT8	✓	✓
CIRCLE		✓
COS	✓	✓
CPOS1, …, CPOS8	✓	✓
CTRL		✓
CTRL_KA		✓
CUBIC_INT		✓
CUBIC_RATE		✓
CUBIC_SCALE		✓
CURR_LIMIT		✓
CURR_OFFSET		✓
CURR_PID		✓
CVEL1, …, CVEL8	✓	✓
DDAC		✓
DELAY		✓
DISABL_INT		✓
DISABL2_INT		✓
ELSE	✓	✓
EN_BUFBRK		✓
ENCOD_MAG		✓
ENDIF	✓	✓
EN_ENCFLT		✓
EN_ERR		✓
EN_ERRHLT		✓
EN_INDEX		✓
EN_MOTCP		✓
EN_POSBRK		✓
EN_PROBE		✓

Table 4-1: DSPL Command Usage Listing

DSPL COMMANDS	PLC	MOTION
ERR1, …, ERR8	✓	✓
ESTOP_ACC		✓
ESTOP_REG	✓	✓
FERR_REG	✓	✓
FERRH_REG	✓	✓
FLUX_CURRENT		✓
FRAC	✓	✓
GEAR		✓
GEAR_OFF		✓
GEAR_OFF_ACC		✓
GEAR_POS		✓
GEAR_PROBE		✓
ICUBCOUNT	✓	✓
IF	✓	✓
INDEX_POS1, …, INDEX_POS8	✓	✓
INDEX_REG	✓	✓
INP1_REG, INP2_REG	✓	✓
INP_STATE		✓
INPUT		✓
INT	✓	✓
INT_HOST	✓	✓
INT_REG_ALL_CLR	✓	✓
INT_REG_CLR	✓	✓
KILIMIT		✓
LINEAR_MOVE		✓
LINEAR_MOVE_S		✓
LINEAR_MOVE_T		✓
LOW_PASS		✓
MAXACC		✓
MOTCP_REG	✓	✓
MOTOR_PAR		✓
MOTOR_TECH		✓
NOTCH		✓
OFFSET		✓
OFFSET_REG	✓	✓
OUTGAIN		✓
OUTP_OFF	✓	✓
OUTP_ON	✓	✓
OVERRIDE	✓	✓
PI	✓	✓
POS1, …, POS8	✓	✓
POSBRK_OUT		✓
POSBRK_REG	✓	✓
POS_PRESET		✓
POS_SHIFT		✓
PROBE_REG	✓	✓
PRINT		✓

Table 4-1 cont.: DSPL Command Usage Listing

DSPL COMMANDS	PLC	MOTION
PRINTS		✓
PROBE_POS1, …, PROBE_POS8	✓	✓
PWM_FREQ		✓
REL_AXMOVE		✓
REL_AXMOVE_S		✓
REL_AXMOVE_T		✓
REL_AXMOVE_SLAVE		✓
RESET		✓
RET		✓
RUN_M_PROGRAM	✓	✓
SIGN	✓	✓
SIN	✓	✓
SINE_OFF	✓	✓
SINE_ON	✓	✓
SQRT	✓	✓
START		✓
STEPPER_ON		✓
STOP		✓
STOP_ALL_M_PROGRAM	✓	✓
STOP_M_PROGRAM	✓	✓
SYNC		✓
TABLE_OFF	✓	✓
TABLE_ON	✓	✓
TABLE_P, TABLE_V	✓	✓
TABLE_SEL		✓
TAN	✓	✓
TIMER, TIMER_RESET	✓	✓
TRQ_LIMIT		✓
VAR1, …, VAR128	✓	✓
VECCHG		✓
VECT4_PAR1, …, VECT4_PAR8	✓	✓
VX4_BLOCK		✓
VEL1, …, VEL8	✓	✓
VELMODE		✓
VIEWVEC		✓
WAIT_UNTIL	✓	✓
WAIT_UNTIL_RTC	✓	✓
WEND	✓	✓
WHILE	✓	✓
=	✓	✓
+	✓	✓
-	✓	✓
*	✓	✓
/	✓	✓
~	✓	✓
&	✓	✓
<, >, <=, >=, ==, !=	✓	✓

Table 4-1 cont.: DSPL Command Usage Listing

The PLC program controls the execution of the Motion programs contained in the DSPL program. The PLC program and up to three Motion programs can be running simultaneously on Mx4.

PLC I/O Functionality

In addition to scanning inputs within the PLC program, the ability to change output status has been added. The `OUTP_ON` and `OUTP_OFF` commands may be used within the PLC program.

PLC Program Syntax

The first line of the PLC program is must be the label `PLC_PROGRAM` followed by a colon (:). The last line of the PLC program must be the keyword `END`.

```
PLC_PROGRAM:

        ;PLC program code here

END
```

PLC Program Examples

Example 1

The following PLC program,

1) initializes two variables, `VAR1` and `VAR2`
2) initializes the Mx4 gains, etc. by running an initialization Motion program
3) initiates execution of TEST_1 Motion program
4) monitors the axis 1 following error, initiating halting procedure if error exceeds limit

```
PLC_PROGRAM:

    VAR1 = 0                      ;initialize variables
    VAR2 = 1

    run_m_program(MX4_INIT)       ;run initialization program
    wait_until(VAR1 == 1)         ;wait for variable condition
    run_m_program(TEST_1)         ;run TEST_1 program
    wait_until(ERR1 > 500)        ;monitor motor 1 error
    run_m_program(HALT_ALL)       ;run halting procedure
END
```

Example 2

PLC programs may initiate simultaneous up to three Motion programs (using Mx4's multi-tasking capabilities) and repeat execution of Motion programs.

```
PLC_PROGRAM:

    VAR1 = 1

    run_m_program (PRG_1,)            ;PRG_1
    while ((CPOS1 > -1)or(CPOS1 < 1)) ;endless while case
        if (VAR1 == 1)
            VAR1 == 0
            run_m_program (EX) ;EX program executed repeatedly
        endif
    wend

END
```

Note : Additional PLC programming examples may be found in the *Applications Notes* chapter.

PLC Program Specifications

Stack Size

Stack size refers to the allowable depth of nested IF-THEN structures in the PLC program. DSPL allows a maximum of 256 IF-THEN constructs in a PLC program.

Writing Motion Programs

What is a Motion Program?

DSPL Motion programs include all of the capabilities of the PLC program in addition to system and motion-related commands. The function of a particular Motion program, thus, is defined by the requirements of an application. The Motion program may emulate PLC monitoring functions or motion commands such as circular and linear interpolations, or a combination of those commands.

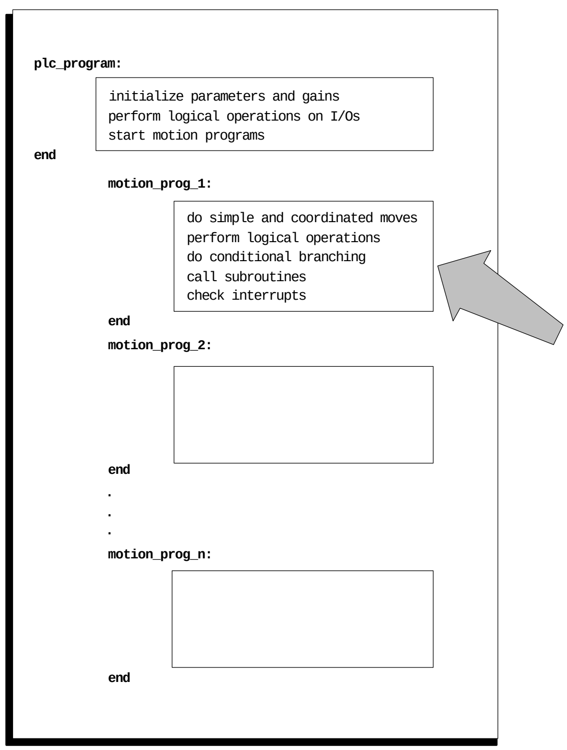


Fig. 4-3: A Typical DSPL Program Sheet, Motion Program Highlighted

The complete DSPL command set is available to Motion programs (see Table 4-1). A DSPL program may contain any number of Motion programs (as opposed to the PLC program, of which only one is permitted). A particular application may require only a single Motion program, whereas the needs of another application may be better served by 20 different Motion programs. The number of Motion programs used in a DSPL program depends both on a particular application and on the programmer's preferences.

In addition to the PLC program, up to three Motion programs can be executed simultaneously on Mx4.

The execution of a motion program is initiated by the `RUN_M_PROGRAM` DSPL command. The execution of a motion program may be terminated by one of the following cases:

- The motion program terminates itself upon reaching the `END` mark of the program
- The DSPL commands `STOP_M_PROGRAM` and `STOP_ALL_M_PROGRAM` will terminate motion program execution
- The host-programming STOP_DSPL RTC will terminate DSPL program execution (and thus any motion programs)

Motion Program Syntax

The first line of a Motion program is its label, up to 21 characters long followed by a colon (:). The last line of this program must be the keyword `END`. For example,

```
CURVE_43DEG:

        ;CURVE_43DEG program code here

END
```

Motion Program Examples

Example 1

The MX4_INT Motion program sets the gains, maximum acceleration, and integral gains limits for axis 1 and axis 4.

```
MX4_INT:

        ;initialize Mx4 parameters

        ctrl (0x9,10,10000,0,2000,0,10000,500,1000)
        maxacc (0x9,0.05,0.13)
        estop_acc (0x9,0.2,0.2)
        kilimit (0x9,2,3)

        VAR1 = 1

END
```

Example 2

The following Motion program performs a simple trapezoidal velocity profile to move motor 3 to target position of 100,000 counts. When the target command position is reached, Mx4 output OUT0 is set. Motion programs can initiate the execution of other Motion programs (similar to the PLC program function) as is included in the TEST example Motion program.

```
TEST:

     axmove (0x4,0.855,100000,3.4)    ;trapezoidal profile
     wait_until (CPOS3 == 100000)     ;wait for end of move
     outp_on (0x0001)                 ;see OUT0
     if (INP1_REG & 0x0200)           ;if input condition is
         run_m_program (TEST2)        ;met, run TEST2
     endif

END
```

Note : Additional Motion programming examples may be found in the *Applications Notes* chapter.

Subroutine Structure

Subroutine calls (up to 15 levels deep) may be made in Motion programs via the `CALL` and `RET` commands. The structure of the subroutine itself is identical to the Motion program structure with the exception that `RET` commands are placed in the subroutine program code to indicate at which point in the subroutine code that the program flow should return to the calling Motion program.

As an example, consider the following subroutine program with three return options,

```
INPUT_CHECK:

        if (INP1_REG & 0x1010)
                VAR3 = 12
                ret ()
        else
                if (INP1_REG & 0x0035)
                        pos_preset (0x4,20000)
                        ret ()
                endif
        endif
        axmove (0x1,0.15,1000,5.0)
        wait (CPOS1 = 1000)
        ret ()

END
```

Motion Program Specifications

Stack Size

Stack size refers to the allowable depth of nested IF-THEN structures in a Motion program. DSPL allows a maximum of 256 IF-THEN constructs in a Motion program.

Using #include files

Many DSPL programs may share similar routines such as Mx4 card initialization routines or emergency motion-halting routines. Rather than copying duplicate Motion programs between DSPL files, the user may wish to use the DSPL compiler `#include` operand. The `#include` operand, when used in a DSPL file, allows the DSPL programmer to link the DSPL file with the specified `#include` file. An `#include` file may contain any number of Motion programs or subroutine codes and, like a DSPL file, must have the *.hll* extension. The `#include` file must be within the same directory as the DSPL file when the DSPL file is compiled.
The correct syntax for the `#include` operand is,

`#include` "*filename.hll*"

The `#include` operand(s) must appear at the beginning of the DSPL file, separate from the PLC and any Motion programming in the file. For example, consider the following DSPL program which includes an `#include` compiler operand,

```
#include "init.hll"

PLC_PROGRAM:

    run_m_program (MX4_INIT)

END
```

where the *init.hll* file consists of,

```
        MX4_INIT:

                ctrl (0x1,10,10000,5000,3400)
                kilimit (0x1,2)
                maxacc (0x1,0.5)
                estop_acc (0x1,1.0)

END
```

Using #define

#define may be used in DSPL programming to customize or personalize `VARx` variable definitions. #define allows the DSPL programmer to assign names to `VARx` variables. For example,

```
#define LENGTHX VAR13
#define toolradius VAR7
```

#defines should be located to the top of the .HLL DSPL text file. References to the variables in the PLC and motion program (s) may use the defined name or the standard `VARx` syntax.

This page intentionally blank.

5 DSPL Basics

Now that you have gained some familiarity with a DSPL program and the PLC and Motion programs which comprise it, let's look at the specific components which make up both PLC and Motion programs. DSPL includes a number of identifiers, operators, and functions.

DSPL Identifiers

- Variables
- Tables
- State Variables
- Input Registers
- Interrupt Registers
- Drive Control Parameters
- Cam & Cubic Spline Table Counters
- Constants

DSPL Operators and Functions

- Basic Arithmetic Operators
- Elementary Math Functions
- Trigonometric Functions
- Relational Operators
- Logical Operators

DSPL Identifiers

The DSPL programming language contains a number of *identifiers*. The DSPL identifiers allow users to:

- Store, retrieve, and modify floating point numbers.
- Create tables.
- Obtain information about system state variables such as position, velocity, and error values.
- Read the status of the Mx4 input registers.
- Check the status of the Mx4 interrupt registers.

Variables

IDENTIFIER	DESCRIPTION
VAR1 to VAR128	General purpose DSPL variables 1 to 128

The DSPL language includes 128 general-purpose variables, which store data in either floating point format for extended precision or as bit registers (when used in bit register operations, see *Bit Register Functionality*). Variables can be used in assignment, function, and relational operations.

```
var3 = var2/var25
var4 = sin(var6)
if (var3 > = var4)
```

Variables can also be used as arguments in DSPL commands. This permits the real time adjustment of motion parameters. For example, the DSPL line:

```
axmove(1, var19, var2, var62)
```

uses variables to perform a real time update of acceleration, slew rate, and target position in a trapezoidal move.

Variables can also be used to store and retrieve data from a table location.

```
table_p(1) = var23
table_v(91) = var11
```

The first line (involving TABLE_P) saves VAR23 in the position format (32-bit value) in the table at location 1. The second line (involving TABLE_V) saves the floating-point value VAR11 in the velocity format (25 bit two's complement value sign extended to 32 bits, the least significant 16 bits represent the fractional value) in the table at location 91. Tables are discussed further in the next section.

Tables

IDENTIFIER	DESCRIPTION
TABLE_P	Mx4 position table. Stores integer values
TABLE_V	Mx4 velocity table. Stores floating point values

DSPL offers 4096 (32-bit) table locations. Table locations can be used to save either integer or fractional values. Integer values (such as positions) can be stored in TABLE_P, while values involving fractions (such as velocities) can be stored in TABLE_V. Numbers in TABLE_P are stored as 32-bit values, while the values in TABLE_V are stored as 25-bit values (sign extended to 32 bits) where the least significant 16 bits represent the fractional portion of the value. The index into the table can be specified as either a constant or a variable. For example:

```
table_p(17) = 42.5
```

saves integer value 42 at index 17. Whereas

```
var50 = 23
table_v(var50) = 42.5
```

will save 42.5 at index 23.

The values to be stored in the table can be specified by either a constant or a variable. Therefore,

```
var49 = 42.5
table_p(17) = var49
table_v(23) = var49
```

will result in the exact same table values as the previous two examples.

Values can also be retrieved from the table. For example, continuing with the previous example:

```
var33 = table_v(23)
```

retrieves the fractional value stored at index 23 of TABLE_V (that is 42.5 if we use the previous example) and stores the value into VAR33. The DSPL instruction:

```
var26 = table_p(17)
```

reads the value stored in index 17 of TABLE_P (i.e. 42 if we continue using the previous examples) in VAR26.

For a slightly more involved example, the DSPL diagram below

```
var3 = 1
while (var3 <= 25)
        table_p(var3) = var3
        var3 = var3 + 1
```

```
wend
```

will save the integer values 1 through 25 in the table locations indexed from 1 to 25. The information saved in the locations indexed from 1 through 25 can be retrieved using the following DSPL code: (note that VAR5 will be overwritten with a new table value each pass through the WHILE structure.)

```
var3 = 1
while (var3 <= 25)
       var5 = table_p(var3)
       var3 = var3 + 1
wend
```

State Variables

IDENTIFIER	DESCRIPTION
CPOS1-8	Command position, axes 1-8
CVEL1-8	Command velocity, axes 1-8
ERR1-8	Following error, axes 1-8
INDEX_POS1-8	Index-capture position, axes 1-8
POS1-8	Actual position, axes 1-8
PROBE_POS1-8	Probe-capture position, axes 1-8
VEL1-8	Actual velocity, axes 1-8

The system state variable values such as position, velocity, and error are available in DSPL as 32-bit registers. The state variables can be used to set the value of a variable. For example:

```
var11 = POS3
```

sets the value of VAR11 to the actual position value of axis 3

State variable can also be used (either alone or in conjunction with variables) in the DSPL conditional structures IF, WHILE, and WAIT_UNTIL. For example,

```
wait_until (POS3 >= var23)
```

prevents execution of the next instruction until the actual position of axis 3 is greater than or equal to the value stored in VAR23.

Input Registers

IDENTIFIER	DESCRIPTION
`ADC1-4`	Analog inputs 1-4
`INP1_REG`	Bit register (IN0 through IN15)
`INP2_REG`	Bit register (IN16 through IN31)

DSPL has two 16-bit input registers, `INP1_REG` and `INP2_REG`, that hold the real time status of the [Mx4:22][Mx4 Octavia:32] external user-defined inputs. The status of the first 16 Mx4 inputs (IN0 through IN15) is contained in `INP1_REG`, while the real time status of the last 16 Mx4 inputs (IN16 – IN31) is held in `INP2_REG`. In both `INP1_REG` and `INP2_REG`, a set bit (bit = 1) indicates an active input condition. Either input register can be used (in conjunction with a bitwise operator) in the DSPL conditional structures `IF`, `WHILE`, and `WAIT_UNTIL`. In the following example:

```
while (inp1_reg & 0x8)
        var12 = 1.5
wend
```

`VAR12` is set to 1.5 only if the signal IN2 is set.

If the Mx4 controller includes the Mx4 Quad ADC Acc4 option, four (4) analog-to-digital (ADC) values are available in DSPL programs. The value (in Volts) that is applied to each of the ADC inputs can be saved in a variable and subsequently transferred to the table. For example, the following command,

```
var23 = ADC3
```

sets `VAR23` to the value (in volts) of the channel 3 voltage. For instance, applying -1.25 volts across the channel 3 input, would result in `VAR23` being set to -1.25 (in floating point format).

Interrupt Registers

IDENTIFIER	DESCRIPTION
`ESTOP_REG`	Bit register signaling ESTOP interrupt
`FERR_REG`	Bit register coding source of following error interrupt
`FERRH_REG`	Bit register coding source of following error /halt int.
`INDEX_REG`	Bit register coding source of index pulse interrupt
`MOTCP_REG`	Bit register coding source of motion complete interrupt
`OFFSET_REG`	Bit register coding source of offset complete interrupt
`POSBRK_REG`	Bit register coding source of pos. breakpoint interrupt
`PROBE_REG`	Bit register coding source of external probe interrupt

The status of a variety of Mx4 interrupt conditions is available to the DSPL programmer via the DSPL interrupt bit registers. All of the DSPL interrupt bit registers, with the exception of `ESTOP_REG`, are 16-bit registers (bit 0-15) that specify the axis(es) responsible for the interrupt. Since there is only one ESTOP signal for all eight (8) axes, `ESTOP_REG` is a single-bit register. In all of the interrupt registers, a set bit (bit = 1) indicates an interrupt.

Like the input registers, interrupt registers can be used (in conjunction with a bitwise operator) in the DSPL conditional structures `IF`, `WHILE` and `WAIT_UNTIL`. For example, the following command:

```
wait_until (index_reg & 0x2)
```

will prevent the execution of the next line until the previously enabled index pulse for axis 2 generates an interrupt. Some or all of the interrupt registers can be cleared by using the DSPL commands `INT_REG_CLR` and `INT_REG_ALL_CLR`.

Drive Control (Vx4++) Parameters

IDENTIFIER	DESCRIPTION
`VECT4_PAR1-8`	`Vx4++` drive control parameters 1-8

When using the Vx4++ option, Vx4++ state variables are available in Mx4s' DSPL programming language. The drive control parameters `VECT4_PAR1` through `VECT8_PAR4` can be assigned one of the following drive variables:

I_{qs}, I_{ds}, I_r, I_s, I_{qs} (feedback), I_{ds} (feedback)

The DSPL command VIEWVEC can be used to determine which one of the above drive variables is assigned to each of the drive control parameters. The following DSPL code:

```
viewvec (0x1,3)
var2 = vect8_par1
```

assign phase current I_s to var2.

Cam and Cubic Spline Table Counter

IDENTIFIER	DESCRIPTION
CAMCOUNT1-8	Cam slave axis table index
ICUBCOUNT	Cubic spline table index

CAMCOUNT1-8 indicates the table index for the slave axes (1-8) engaged in camming.

The users utilizing Mx4's internal cubic command can benefit from the ICUBCOUNT counter. This DSPL reserved word is used in conjunction with cubic spline instructions and indicates the active cubic spline table index.

Constants

IDENTIFIER	DESCRIPTION
PI	Approximation to π (3.14159265)

The DSPL constant PI is a reserved word that can be used in arithmetic, trigonometric, and conditional expressions as an approximation to the value π (3.14159265).

```
var1 = pi/2
var2 = cos(pi)
```

Timer

The keyword TIMER on Mx4/Mx42 (TIMER1, TIMER2, TIMER3, and TIMER4 on Octavia) may be read into a variable or used in conditional statements such as IF, WHILE, or WAIT_UNTIL. The timer units are 200μsec. The timer may be reset with the `TIMER_RESET()` command. Note that the timer is always running, and that the `TIMER_RESET()` command will reset the timer value to 0.

For example, to turn on outputs 0, 1, and 2 in succession 750msec apart, the following Mx4/Mx42 code is used.

```
TIMER_RESET ()
OUTP_ON (0x0001)
WAIT_UNTIL (TIMER >= 3750)
OUTP_ON (0x0002)
WAIT_UNTIL (TIMER >= 7500)
OUTP_ON (0x0004)
```

DSPL Operators and Functions

The DSPL operators and functions act on either one or two of the DSPL identifiers. A sample DSPL program using its operators and functions is shown below:

```
var1 = -1.92
var2 = 3.285e+003
var3 = var1/var2              ;var3 is set to
                              ;0.0005916795069
var8 = abs(var1)              ;var8 is set to 1.92
var5 = sqrt(var2)             ;var5 is set to 56.9689015
var6 = sin(2.1)               ;var6 is set to 0.863209366
```

The following sections briefly describe each of the operators and functions.

Basic Arithmetic Operators

OPERATOR	DESCRIPTION
=	Assignment
+	Addition
-	Subtraction
*	Multiplication
/	Division

The assignment "=" operator is the simplest of the DSPL operators, and can be used to set the value of a variable or a table entry equal to a constant value. For example:

```
var1 = -1.92
var2 = 3.285e+003
```

```
var3 = 0x38           ;var3 is set to a hexadecimal number
value 38 = 52
```

The assignment operator can also be used to set the value of a variable equal to the result of an arithmetic operation. For example:

```
var1 = -1.92
var2 = 3.285e+003
var3 = var1 + 11.1            ;var3 is set to 9.18
var8 = var1/var2              ;var8 is set to -
                              ;0.0005916795069
```

Elementary Math Functions

FUNCTION	DESCRIPTION
`ABS( )`	Absolute value
`FRAC( )`	Fraction function
`INT( )`	Integer function
`SIGN( )`	Sign function
`SQRT( )`	Square root function

The elementary math functions work on a single variable or constant value. The examples in this section continue the example in the previous section.

The function `ABS( )` finds the absolute value of a constant or a variable value.

```
var5 = abs(var1)              ;var5 is set to 1.92
```

The function `FRAC( )` extracts the fractional portion of a constant or a variable value.

```
var6 = frac(var1)             ;var6 is set to -0.92
```

The function `INT( )` extracts the integer portion of a constant or a variable value.

```
var7 = int(var1)              ;var7 is set to -1
```

The function `SIGN( )` returns +1, 0 or -1 depending on whether a constant or a variable value is greater than, equal to, or less than 0.

```
var8 = sign(var1)             ;var8 is set to -1
```

The function `SQRT( )` calculates the square root of a constant or a variable value.

```
var9 = sqrt(var2)              ;var9 is set to 56.9689015
```

Trigonometric Functions

FUNCTION	DESCRIPTION
`ARCTAN( )`	Arctangent function
`COS( )`	Cosine function
`SIN( )`	Sine function
`TAN( )`	Tangent function

Trigonometric functions work on either constant or variable values. The arguments in the functions `SIN`, `COS`, and `TAN` are expressed in radians. The result of `ARCTAN` is expressed in radians.

```
var1 = 1.5707
var3 = sin(var1)               ;var3 is set to 0.99999999
var8 = cos(var1)               ;var8 is set to 0.000096326
var5 = arctan(var1)            ;var5 is set to 1.00805632
```

Relational Operators

OPERATOR	DESCRIPTION
`<`	Less than
`>`	Greater than
`<=`	Less than or equal to
`>=`	Greater than or equal to
`==`	Equal to
`!=`	Not equal to

Relational operators are used in conditional statements in the DSPL conditional structures `IF`, `WHILE` and `WAIT_UNTIL`. For example:

```
wait_until(POS1 >= 38)
```

will prevent execution of the next instruction until the actual position of the first axis (i.e. `POS1`) is greater than or equal to 38 counts.

Bitwise and Logical Operators

OPERATOR	DESCRIPTION
`~`	Bitwise complement
`&`	Bitwise AND
`AND`	Logical AND
`OR`	Logical OR

Bitwise and logical operators are used with both input and interrupt registers in conditional expressions. The bitwise operator “&” is used for masking a selected number of bits in an input or interrupt register. The bitwise operator “**~**“ complements the contents of a register. Logical operators `AND/OR` work on the conditional statements in the DSPL conditional structures `IF`, `WHILE`, and `WAIT_UNTIL`. For example, the DSPL conditional expression line below:

```
if ((inp1_reg & 0x3) AND (~inp2_reg & 0x1))
```

will first mask all but the two least significant bits of input register 1, then mask all but the least significant bit of the complemented input register 2, and finally perform a logical `AND` of the results. For a bitwise condition to be true, there must be an exact match between set bits in the mask and corresponding bits of the register (or ~register).

Bit Register Functionality

Bit Register Functionality enables variables to be manipulated as 16-bit bit registers. Specifically, the following bit register operations are available.

- `VAR[1-128]` = hex constant

  for example,
  ```
  VAR41 = 0xA055
  ```

- `VAR[1-128]` = bit register (registers ending with _reg, such as `inp1_reg`)

  for example,
  ```
  VAR33 = INP2_REG
  VAR15 = MOTCP_REG
  ```

- `VAR[1-128] = VAR[1-128] &` 16-bit mask

  for example,
  ```
  VAR1 = VAR1 & 0x00FF
  VAR12 = VAR12 & 0x0003
  ```

- `VAR[1-128] = VAR[1-128] |` 16-bit mask

  for example,
  ```
  VAR51 = VAR3 | 0xFF00
  VAR2 = VAR2 | 0x0001
  ```

- `VAR[1-128] = VAR[1-128] & VAR[1-128]`

  for example,
  ```
  VAR1 = VAR1 & VAR44
  VAR12 = VAR12 & VAR1
  ```

- `VAR[1-128] = VAR[1-128] | VAR[1-128]`

  for example,
  ```
  VAR21 = VAR3 | VAR15
  VAR8 = VAR72 | VAR82
  ```

- `VAR[1-128] = ~ VAR[1-128]` bitwise complement

  for example,
  ```
  VAR59 = ~ VAR3
  VAR24 = ~ VAR8
  ```

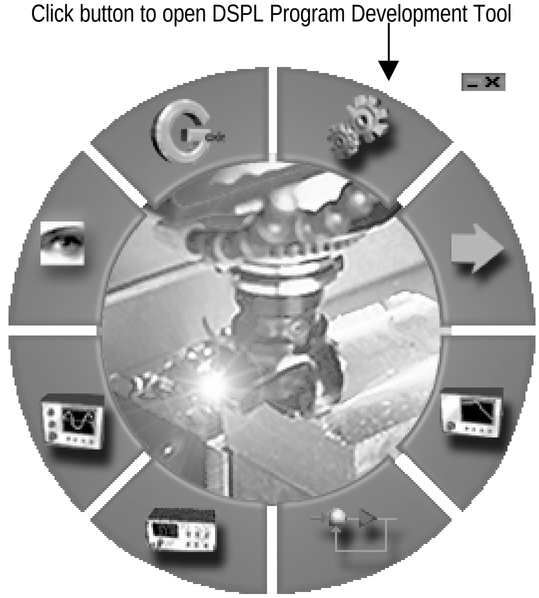
- Logical condition checks for `IF`, `WAIT_UNTIL`, `WHILE`
  `VAR[1-128] &` 16-bit mask
  `VAR[1-128] |` 16-bit mask
  `~VAR[1-128] &` 16-bit mask
  `~VAR[1-128] |` 16-bit mask

  for example,
  ```
  WAIT_UNTIL(VAR24 & 0x0010)
  WHILE(~VAR1 & 0x0001)
  ```

6 DSPL Program Development

Note: This chapter assumes prior installation of the Mx4Pro Development Tools v5.x (see the *Mx4Pro Development Tools* manual, Chapters 2 and 3).

Click button to open DSPL Program Development Tool



The DSPL Program Development Tool allows you to create, modify, compile, download, and execute DSPL programs. The DSPL Development Tool may be opened by clicking on the **DSPL** button on the Mx4Pro Development Tools tool bar. The following window will appear:

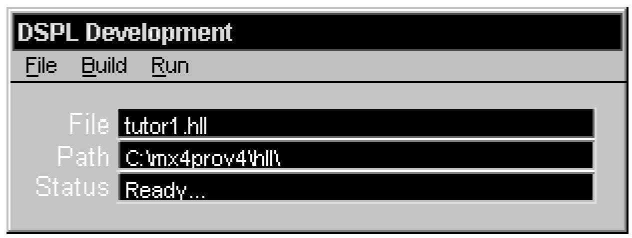


Fig. 6-1:   DSPL Development Tool

The DSPL Development tool displays the name of the open DSPL **File**, the **Path** for the open file, and the **Status** of a compile and/or download performed on the file. The following sections describe how to utilize the different features of the DSPL Development Tool.

Opening DSPL Files

Before a DSPL program can be edited, compiled, or downloaded, it must be opened by the user. To open a DSPL program:

1. **Open the DSPL Selection window**. This can be achieved by selecting **Open...** under the **File** menu or in the popup menu (right click in the DSPL Development window). Double-clicking inside one of the three black areas (File, Path, or Status text boxes) inside the DSPL Development window (Figure 4-1) will also open the DSPL Selection window.
2. **Select the DSPL File**. To open a file, browse your hard drive to the path where your project will exist or does exist, then click on the filename or enter the file name in the **File Name** text box. Note, there is a **File Type** filter. After your file has been selected, click on the **OK** button to accept your selection or the **Cancel** button to disregard (Figure 6-2).

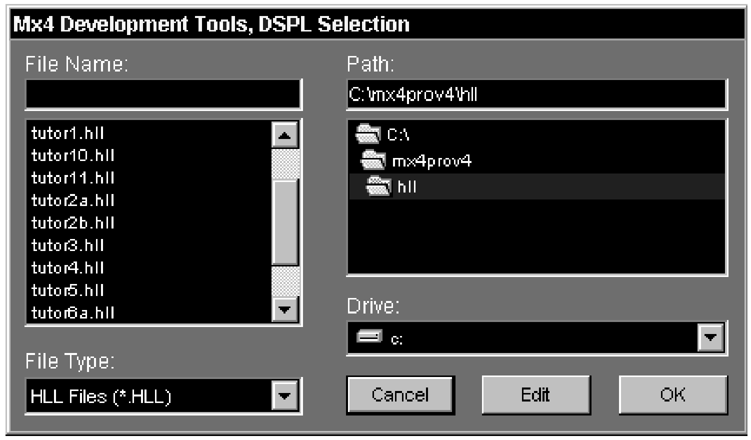


Fig. 6-2: DSPL Selection window

Editing Files

After a file has been opened, it may be edited. This can be achieved by selecting **Edit** under the **File** menu or in the popup menu. The opened file will then be placed into a text editor. To select the editor used for editing the DSPL programs, refer to *Mx4Pro Development Tools* "Selecting an Editor" in Chapter 12, Advanced Topics.

A file that has not been opened may also be edited via the DSPL Selection window. Follow steps 1 and 2 above, then select the **Edit** button instead of the **OK** button (Figure 6-2). The file will then be opened with the editor, but not into the DSPL Development tool. This feature is useful when an "include" file needs to be edited or created.

Compiling Files

An opened DSPL file can be compiled using the DSPL Development tool, but make sure you save the file first, if it has been edited. To compile the DSPL file, select **Compile** under the **Build** menu or in the popup menu. For example, by selecting **Compile** from within the popup menu, the DSPL compiler will compile the opened DSPL program (Figure 6-3).

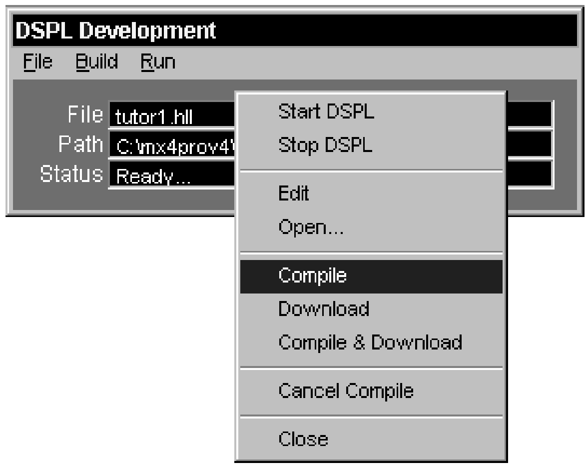


Fig. 6-3 Compiling DSPL Program

After the compile has started, it may be canceled by selecting **Cancel Compile** under the **Build** menu.

If the DSPL compiler detects any warnings or errors during the compilation of the opened file, the **Status** box in the DSPL Development Tool will display a warning/error message and an edit session displaying the warnings and/or errors will appear.

Downloading Files

If the opened file has been compiled successfully, it can be downloaded to the Mx4 card by selecting **Download** under the **Build** menu or in the popup menu.

The opened DSPL file may also be compiled and downloaded if **Compile and Download** is selected under the **Build** menu or in the popup menu.

Note: If **Compile and Download** was used and any warning(s) and/or error(s) occurred, then the file will NOT be downloaded. If only warnings were issued the file may still be downloaded, but **Download** must be used instead of **Compile and Download**.

Executing DSPL Programs

There are several commands that may be issued to control the execution of a downloaded DSPL program. The following commands may be issued to the Mx4 via the DSPL Development Tool by selecting the appropriate command under the **Run** menu,

- **Start DSPL -** Starts the DSPL program execution
- **Stop DSPL -** Stops the DSPL program execution
- **Signal DSPL -** Signals the DSPL program, breaks out of a WAIT_UNTIL_RTC command in a DSPL program.
- **AutoStart DSPL -** Select **AutoStart On** or **AutoStart Off** to turn the autostart option on or off, respectively.

A DSPL program may also be started or stopped by selecting **Start DSPL** or **Stop DSPL**, respectively, from within the popup menu. Furthermore, the function keys F1 through F3 may be used to issue the **Start DSPL** (F1), **Stop DSPL** (F2), and **Signal DSPL** (F3) commands when the DSPL Development Window is active.

Refer to the *Mx4 User's Guide* for more information on these commands.

Monitoring Execution of DSPL Program

The execution and run-time status of a DSPL program may be monitored by a host computer. The line number of the PLC program and three motion programs that are currently executing is available in the Mx4 Dual Port RAM DSPL updates window (066h - 085h). DSPL run-time errors are reported to the Mx4 DPR DSPSTAT2 (009h) status register.

Closing the DSPL Development Tool

To close the Mx4Pro Development Tool, select **Close** under either the **File** menu or in the popup menu. The opened DSPL file and path along with the window dimensions and position are saved. When the window is started again the same DSPL file will be opened and the window will appear in the same location as when it was closed.

7 Tutorial

Now that you have seen DSPL and the constructs, keywords, commands and identifiers which make up DSPL application programs, you're ready to start your own DSPL programming. The following tutorials illustrate different functionalities of the DSPL language in working examples which may be complied, downloaded, and executed with the Mx4pro DSPL Program Development Tool (see chapter 6, *DSPL Program Development*). The following tutorial DSPL files are located in the HLL folder and any referenced data files are located in the DAT folder of the Mx4Pro install directory.

Session 1 Getting Started

As you know, every DSPL program needs a section entitled `PLC_PROGRAM`. The PLC program includes calls to motion programs as well as Boolean operations such as `IF`, `WHILE`, and `WAIT_UNTIL`. For example, in the following program the only function which the PLC performs is starting the execution of the motion program "my_first". Immediately following the start of execution of the "my_first" motion program, the PLC execution terminates as the `end` line command is reached. The "my_first" execution continues, however, until the `end` line command in the "my_first" motion program is reached.

```
plc_program:
    run_m_program (my_first)
end

my_first:
    pos_preset(1,0)                   ;set position of axis 1 to 0
    ctrl (1, 0, 2000, 1000, 1000)     ;set control gains for axis 1
    axmove (1, 1, 20000, 5)           ;move axis 1 to location 20000
end
```

Remember, this tutorial example program, tutor1.hll, as well as the examples from sessions 2 through 11 are included with the *Mx4 pro Development Tools* software.

The first line of motion program, “my_first”, clears any error, and presets the axis 1 position counter to a value of 0. The next line contains the control gain settings ki=0, kp=2000, kd=1000, and kf=1000 for axis 1. If the Mx4 controller is already connected to your system, you must make sure that the control gains have been optimally selected. The next line, `AXMOVE`, specifies acceleration, target position, and traveling speed for a trapezoidal move. This simple program simply presets the current position, closes the loop by setting control law parameters, and moves axis 1 to position location 20000.

Session 2 Using Variables

In this session you will learn how to:

- Use variables as arguments in DSPL commands
- Use variables in mathematical expressions.

DSPL variables are used for real-time computation of system dynamics. The arithmetic and geometric operators are used in conjunction with variables, allowing application programs to compute motion parameters “on the fly”. The following shows an example (tutor2a.hll) of a system in velocity mode.

```
plc_program
    run_m_program(var_speed)
end

var_speed:
    ctrl (1, 0, 2000, 1000, 1000)    ;set control gains for axis 1
    maxacc(1,1)                      ;set maximum acceleration for
                                     ;axis 1 to 1 count/200usec^2
    pos_preset(1,0)                  ;preset position of axis 1
    var1 = 0
    while (var1 <= 1000)
        var2 = 0.01*var1
        var23 = sin(var2)            ;compute a sinusoidal command
        velmode (1, var23)           ;use var23 for axis 1 speed
    wend
end
```

The tutor2a.hll program runs axis 1 at a constant speed, as the var1 variable value is not changed, and program calculations yield a constant value.

The same program may be modified to run axis 1 at a variable speed determined by an arbitrary equation. In the following example (turor2b.hll) we use trigonometric function `SIN` to change the speed sinusoidally.

```
plc_program
    run_m_program(var_speed)
end

var_speed:
    ctrl (1, 0, 2000, 1000, 1000)     ;set control gains for axis 1
    maxacc(1,1)                       ;set maximum acceleration for
                                      ;axis 1 to 1 count/200usec^2
    pos_preset(1,0)                   ;preset position of axis 1
    var1 = 0
    while (var1 <= 1000)
        var2 = 0.01*var1
        var23 = sin(var2)             ;compute a sinusoidal command
        velmode (1, var23)            ;use var23 for axis 1 speed
        var1 = var1 - 1               ;decrement var1
    wend
end
```

Session 3 Mathematical Functions

In this session you will learn about:

- Using DSPL arithmetic functions
- Using DSPL trigonometric functions

The arithmetic functions and mathematical operators are used in conjunction with real-time computation of arguments used in DSPL instructions. The following example describes how the trigonometric expression:

$$1000 * (1 - \cos(2\pi t/T))$$

is computed. Also, this example (tutor3.hll) shows how the results are saved in a table array.

```
math:
    var2 = 0
    var10 = 0                         ;var10 indexes through table
    var3 = 25                         ;var3 holds the period in ms
    while  (var10 <= var3)            ;compute expression from 0 to T
        var4 = 2*pi
        var5 = var4/var3              ;compute 2π/T
        var7 = var5 * var10           ;2πt/T
        var8 = cos(var7)
        var9 = 1 - var9               ;1-cos(2*π*t/T))
        var9 = 1000*var9              ;1000*(1-cos(2*π*t/T))
        table_p(var10) = var9         ;save values in consecutive
```

```
        var10 = var10 + 1             ;table locations
    wend
ret()
```

The main program may access an element of the saved table array via a DSPL line such as:

```
    var25 = table_p(3)
```

which simply reads location 3 of the table into var25. For more information on arithmetic and trigonometric functions please refer to the command descriptions for the following commands (chapter 8, *DSPL Command Set*):

```
ARCTAN        COS           SIN
SQRT          TAN           TABLE_P
TABLE_V       +, -, *, /
```

Session 4 Electronic Gearing

This tutorial illustrates the use of electronic gearing, as the following example describes a packaging process that includes two conveyor belts. The upper belt contains products which are equally positioned in between the logs. The master motor moves the products and drops them into the bucket. The synchronization between the belts requires gearing mechanism. The gear ratio in this example is determined by the ratio of the space between the centers of the adjacent buckets and the space between the products. The following program, (tutor4.hll) upon setting a “start switch,” puts the system in electronic gearing and drives the master axis at a constant speed of 4 counts/200 µs. Upon pushing a “stop switch,” the system terminates gearing and comes to a halt.

```
plc_program:
    run_m_program(simple_gear)
end

simple_gear:
    maxacc(0x3,1,1)            ;set maximum acceleration for stop
    ctrl(0x3,0,1000,1000,1000,0,1000,1000,1000)
                               ;set control gains for master and slave
    wait_until(inp1_reg & 0x0001)
                               ;wait for “start” switch, Mx4 IN0
    gear(1,2,2)                ;master axis is 1, slave axis is 2,
                               ;and gear ratio is 2
    velmode(1,4)               ;move master at constant speed of 4
    wait_until(inp1_reg & 0x0002)
                               ;wait for the ‘stop’ switch
```

```
    stop(1)                    ;stop the master
    gear_off_acc(2)            ;stop slave and disengage the gear
end
```

As is the case with most DSPL commands, the arguments used in conjunction with electronic gearing may be selected as DSPL variables.

Session 5 Cam Programming

In this session, through two examples you will learn how to:

- Fill the Mx4 memories with cam points (i.e. master/slave positions) either off-line or on-line
- Write a DSPL program to perform camming

Example 1: Cam Program, Using Host to Download Positions

Consider a table of 10 master/slave position points for x (master) and y (slave) formed as follows:

Master	**Slave**
0	0
1000	200
1500	400
2000	600
2500	700
3000	800
3500	600
4000	400
4500	200
5000	0

This table can be saved in an ASCII data file (with .dat extension) under any name (e.g. cam_tut5.dat). Using the Mx4pro cam table download utility, you may download this file starting at any cam table index.

The following DSPL program will perform the cam function on axis 1 (the master) and axis 2 (the slave).

```
plc_program:
    run_m_program (simple_cam)
end

simple_cam:
    ;*****************************************************
    ;
    ;   In this example, we assume that you have used
    ;   the cam download utility included in Mx4pro, and
    ;   have downloaded "cam_tut5.dat" which includes 10
    ;   master/slave cam points into the Mx4 data memory
    ;
    ;*****************************************************
    ctrl (0x3,0,1000,1000,1000,1000,10000,5000,3000)
    maxacc (0x3,1,1)          ;set maximum accel
    pos_preset(0x3,0,0)       ;preset xy positions
    velmode(1,5)              ;run master in velocity mode
    cam(1,2,100,10)           ;start cam function
end
```

Example 2: Cam Program, Using DSPL To Generate the Cam Points in Real-Time

This is similar to example 1 with the exception that the cam points have been defined (it is important to remember that they might have been computed) by the DSPL program using the `CAM_POINT` command.

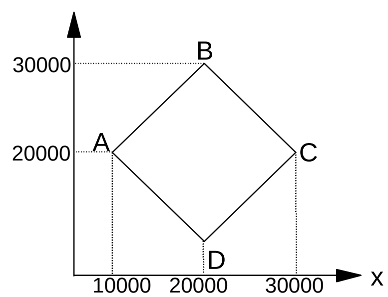
```
plc_program:
    run_m_program (simple_cam)
end

simple_cam:
    ;*****************************************************
    ;
    ;    In this example, 10 cam points specified
    ;    by master and slave positions are defined by
    ;    the DSPL and put in the Mx4 cam memory.
    ;    Master is axis 1, slave is axis 2.
    ;    Master starts in velocity mode. This is
    ;    followed by running cam function
    ;
    ;*****************************************************
```

Session 6 Linear Moves

DSPL includes two forms of linear interpolated motion:

```
Linear_move_s          ;s-curve, acceleration
linear_move            ;constant acceleration
```

The linear motion commands are used in motions where the velocity connecting point A to Point B is linear. The starting position/velocity (defining point A) are those of an axis at the commencement of this command. The ending position and velocity are the command's arguments. The following example (tutor6a.hll) will trace a square shape as illustrated below.




```
plc_program:
    run_m_program (square)
end_program

square:
    var23=1
    ctrl(0x3,0,1000,1000,1000,0,1000,1000,1000)
                              ;set control gains for motor 1
    pos_preset(0x3,10000,20000)              ;point A
    while(var23==1)

        linear_move(0x3,15000,5,25000,5)             ;point AB/2
        linear_move(0x3,20000,0,30000,0)             ;point B

        linear_move(0x3,25000,5,25000,-5)            ;point BC/2
        linear_move(0x3,30000,0,20000,0)             ;point C

        linear_move(0x3,25000,-5,15000,-5)           ;point CD/2
        linear_move(0x3,20000,0,10000,0)             ;point D

        linear_move(0x3,15000,-5,15000,5)            ;point DA/2
        linear_move(0x3,10000,0,20000,0)             ;point A

    wend
end
```

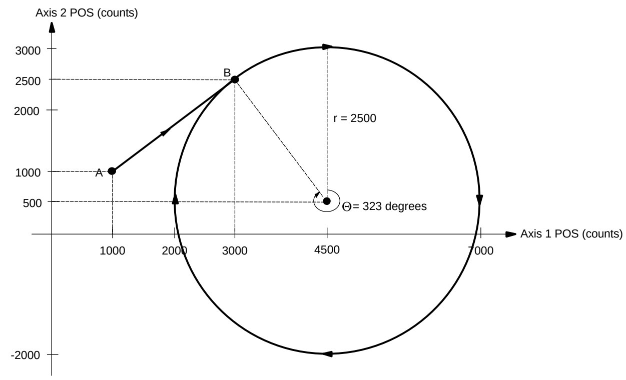
A slightly more involved linear move is one in which the velocity profile is an "s-curve" (i.e. jerk is programmable). The following program (tutor6b.hll) moves axes 1 and 2 in a coordinated move from the initial position (1000, 1000) counts and velocity (0,0) counts/200 µs to the target position (3000, 2500) and velocity (0.8, 0.6).

```
plc_program
    run_m_program (line)
end_program

line:
    ctrl (0x3,0,1000,1000,1000,0,1000,1000,1000)
                                        ;set the gains
    pos_preset (0x3, 1000,1000)         ;preset the pos command
    linear_move_s (0x3, 1000,0 30000, 0.8, 5000, 0.0003,
    1000,0,2500,0.6,5000,0.00022)
end
```

Session 7 Circular Moves

An example of a circular move can be generated by the following code:

```
plc_program
    run_m_program (circle_move)
end_program

circle_move:
    linear_move_s(0x3,1000,0,2500,0.8,5000,0.0003,1000,0,3000,0.6,
     5000,0.0003)
    circle(0x3,1500,-2000,2500,-3,0,0)
end
```



The `LINEAR_MOVE_S` arguments used in this example are initial position and velocity for x (1000,0), final position and velocity for x (3000, 0.8), time to complete x motion (5000), x acceleration value during constant acceleration segment (0.0003), initial position and velocity for y (1000, 0), final position and velocity for y (2500, 0.6), time to complete y (5000), y acceleration.

The arguments for `CIRCLE` command are: the x-y values for its center ($cent_x$ = 4500 - 3000 = 1500, $cent_y$ = 500 - 2500 = -2000), radius (sqrt $((2000)^2 + (1500)^2) = 2500$), vector speed (1.0), and target position for x and y (x = 3000 - 3000 = 0, y = 2500 - 2500).

Session 8 Table-Based Cubic Spline

In this session you will learn how to:

- Form a cubic spline table and download it to the Mx4
- Write a DSPL program to use a cubic spline data file

1. Generate and Down Load A Cubic Spline Table

Consider a simple application in which the x axis position and velocity are tabulated as follows:

```
posx1          velx1
posx2          velx2
.              .
.              .
posxn          velyn
.
.
```

The position and velocity points are in encoder counts and encoder counts/s respectively. Also, adjacent positions are spaced in time uniformly. For example, the following ASCII data file (cub_tut8.dat) includes 21 rows of position and velocity for axis x.

```
0.00000000000000e+000 0.000e+000
1.25000000000000e+003 2.5000e+004
5.00000000000000e+003 5.0000e+004
1.00000000000000e+004 5.0000e+004
```

```
1.50000000000000e+004 5.0000e+004
2.00000000000000e+004 5.0000e+004
2.50000000000000e+004 5.0000e+004
3.00000000000000e+004 5.0000e+004
3.50000000000000e+004 5.0000e+004
3.87500000000000e+004 2.5000e+004
4.00000000000000e+004 0.0000e+004
3.87500000000000e+004 -2.5000e+004
3.50000000000000e+004 -5.0000e+004
3.00000000000000e+004 -5.0000e+004
2.50000000000000e+004 -5.0000e+004
2.00000000000000e+004 -5.0000e+004
1.50000000000000e+004 -5.0000e+004
1.00000000000000e+004 -5.0000e+004
5.00000000000000e+003 -5.0000e+004
1.25000000000000e+003 -2.5000e+004
0.00000000000000e+000 0.0000e+000
```

The Mx4pro Development Tools software may be used to download this file using the Tables, Cubic Spline menu option.

2. Write a DSPL Program to Use This Data Array

The following DSPL program (tutor8.hll) will run our previously generated data array:

```
plc_program:

    run_m_program (cubic)

END

cubic:
    ctrl(1,0,1000,1000,1000)      ;set the gains
    pos_preset(0x1,0)             ;preset the command
    cubic_rate(500)               ;time interval between adjacent
                                  ;points is 100 ms
    cubic_scale(0x1,1.5,0)        ;scale table values by 1.5 and
                                  ;include no shift
    cubic_int(21,0,3)             ;starting from location 0 (middle
                                  ;argument) 21 points to run 3
                                  ;times
end
```

For more information on cubic spline and its use, please refer to *Cubic Spline Application Notes.*

Session 9 ASCII Terminal Communication

With the Acc4 serial communication option, the Mx4 controller includes an ASCII terminal serial interface which includes ASCII terminal commands as well as ASCII DSPL commands.

The ASCII terminal commands enable the ASCII terminal user to both read and write DSPL variables. The reading and writing of the 128 DSPL variables `(VAR1-VAR128)` is done independently of the DSPL program execution. Variable values can be queried during DSPL program execution to monitor state variables or other program parameters of interest. Also, the ASCII terminal interface allows users to set DSPL program parameters and control DSPL program flow from the terminal by writing variable values which are utilized within the DSPL program.

The ASCII DSPL commands allow an executing DSPL program to write values and character strings to the ASCII terminal display as well as ‘input’ values sent from the ASCII terminal to DSPL variables.

The Mx4 controller can communicate via two (2) different serial modes: ASCII mode and Protocol mode. The Protocol mode is the ‘standard’ mode of communication supported by Mx4 family utilities such as the Mx4pro development tools. The Protocol mode supports faster data rates with multilayer error detection and correction for industrial environments.

The ASCII mode of communication is, as the name implies, for users who would like to use an ASCII terminal for some basic information passing to the Mx4 controller; that is, reading and writing DSPL variables.

ASCII Mode Terminal Commands

The ASCII mode of communication supports four (4) terminal commands,

`EC0`

Echo Off. The `EC0` command turns the echo mode off. The Mx4 upon power-up or reset is in the `EC0` or echo off mode.

`EC1`

Echo On. The `EC1` command turns the echo mode on. The Mx4 upon power-up or reset is in the `EC0` or echo off mode.

`VARx?`

Read DSPL Variable. This command queries the specified DSPL variable (x : 1 to 128). The value displayed is an integer with 3 implied fractional digits. For example, 123456 is the value 123.456.

`VARx=y`

Write DSPL Variable. This command writes the value `y (-2147000000<=y<=2147000000)` to the specified DSPL variable `(x : 1 to 128)`. The value written is an integer with 3 implied fractional digits. For example `VAR12=123456` will set `VAR12 to 123.456`.

ASCII Mode DSPL Commands

The ASCII mode of communication supports three (3) Mx4 DSPL commands,

```
PRINT, PRINTS, AND INPUT
```

The PRINT command is used to write (send) a value to the ASCII terminal display. The ASCII transmission to the terminal takes the format:

```
(value) + <CR> + <LF> + ‘>’
```

The value displayed is an integer with 3 implied fractional digits. For example, 123456 is the value 123.456.

For example, to write the value 100.45 to the ASCII terminal:

```
PRINT (100450)
```

To write the value contained in DSPL variable `VAR128` to the ASCII terminal

```
PRINT (VAR128)
```

The PRINTS command is used to write (send) a character string to the ASCII terminal display. The ASCII transmission to the terminal takes the format:

```
(string) + <CR> + <LF> + '>'
```

For example, write "hello world" to the ASCII terminal.

```
PRINTS ("hello world")
```

The INPUT command is used to write a value sent by the ASCII terminal to the specified DSPL variable. The ASCII transmission to the terminal takes the format:

```
'??'
```

The DSPL motion program from which the INPUT command was executed will halt (wait) program execution until the value is returned from the ASCII terminal. The ASCII transmission from the terminal to the Mx4 must follow the format:

```
Inp=x
```

Where x may range from `–2147000000 <= x <= 2147000000`. The value written is an integer with 3 implied fractional digits. For example, `inp=123456` will set the specified variable to `123.456`.

For example, request ASCII input, assign to VAR15.

```
INPUT (VAR15)
```

Session 10 Vector Control

In this session you will learn:

- Programming Vx4++ parameters with a `#include` file
- Reading Vx4++ state variables in a DSPL program

When using the Vx4++ option, the user must program current loop parameters in addition to the position loop initializations and gains. As the number of parameters which must be initialized grows, the user may wish to incorporate the `#include` DSPL compiling option. With the `#include` feature, the user may link in common routines such as initialization and/or emergency halting routines which exist in separate DSPL .hll files.

The following DSPL program (tutor10.hll) utilizes the `#include` feature to link in the file init10.hll. Included in the init.hll file is the initialization motion program INIT_V4. The Mx4/Vx4++ initialization is performed with the subroutine call to INIT_V4.

```
#include "init10.hll"
plc_program:
    run_m_program(test_v4)
end

test_v4:
    var1 = 0
    call(init_v4)              ;init_v4 is in the #include file
                               ;init.hll
    wait_until(var1 == 1)      ;var1 is a flag to let the main
                               ;program know it is done initializing
    viewvec (0x1, 3)           ;specify that the axis 1 Vx4++ state
                               ;variable is Ids feedback
    pos_preset ( .... )        ;code as required by application
    axmove ( .... )
    etc., etc.

    if (vect4_par1 > 1250)     ;the vect4_par1 is the state variable
        flux_current (0x1, 12) ;specified in the viewvec command ...
    endif                      ;Ids feedback

    .
    .
    .

end
```

The init10.hll file contains the "init_v4" motion program which initializes the system parameters,

```
init_v4:
    maxacc (0x1,1.9)
    ctrl (0x1,0,8632,912,560) ;initialize position loop gains (Mx4)
    pos_preset (0x1,0) ;initialize current loop parms.

                                      ;(Vx4++)
    motor_tech (0x1,brushless_dc)     ;brushless DC
    motor_par (0x1,0)                 ;motor parameter is 0
    curr_limit (0x1,30)               ;set current limit at 30%
    curr_offset (0x1,800)             ;set offset to 800
    curr_pid (0x1,30000,0,3000)       ;current loop pid gains
    encod_mag (0x1,1000,4,1)          ;1000 lines, 4 poles, and comm 1
    flux_current (0x1,9)              ;field command set to 9
    pwm_freq (0x1,15000)              ;set pwm frequency to 15 khz

    var1=1
    ret()
end
```

You may have noticed that the above listed DSPL program includes a `VIEWVEC` command call. The `VIEWVEC` is used (in a DSPL program) in conjunction with the `VECT4_PARx` state variable identifiers. The `VIEWVEC` command specifies the Vx4++ state variables which are represented by the DSPL `VECT4_PARx` identifiers. In the example program, the axis 1 Vx4++ state variable is defined as I_{ds} feedback. Subsequent uses of the `VECT4_PAR1` identifier throughout the program are referencing the Ids feedback state variable. For example, note the `IF` code in the example program which utilizes the `VECT4_PAR1` identifier.

Session 11 Using Interrupts

In this section you will learn about:

- DSPL Interrupts, and
- How they are used, disabled, and cleared

The DSPL interrupts are used when an immediate reaction to an external event is required. An example application is mark registration. In this application, the motor position is corrected by the amount measured at the time of receiving an interrupt. The external pulse which, for instance, is originated from an electronic eye, must be hardwired to a Mx4 interrupt (e.g. EXT1). The instruction `EN_PROBE` enables this interrupt.

A typical DSPL program (tutor11.hll) for this application is as follows:

```
plc_program:
    run_m_program(ptest)
end

ptest:
    ctrl(1,0,1000,1000,1000)  ;set control gains
    pos_preset(1,0)           ;preset the position of axis 1 to 0
    int_reg_all_clr()         ;clear all interrupt registers
    en_probe(0x1)                     ;enable EXT1,stop when EXT1 is set
    velmode (1,3)                     ;run axis one at 3 c/200 µ
    wait_until(probe_reg & 0x1)       ;wait for the probe
                                      ;i.e.EXT1)interrupt
    delay (10000)                     ;wait until the axis comes to stop
    var4 = probe_pos1 - pos1          ;find the difference between
                                      ;current pos and EXT1 position
    rel_axmove(1,1,var4,5)            ;move the axis back to probe
                                      ;location at 5 c/200 µs speed
    end
```

Similarly, you may use this technique in “homing” an axis where the reference position is determined by the location of Index pulse.

Other interrupts which may be enabled in a DSPL program are:

```
en_encflt      encoder fault
en_err         error exceeding a programmed value
en_errhlt      stop when error exceeds a programmed value
en_index       occurrence of index pulse
en_motcp       motion complete
```

```
en_posbrk      position break point
```

Interrupts may be disabled or cleared by the commands:

```
disabl_int             disable interrupts
disable2_int           disable interrupts
int_reg_all_clr        clear all interrupt registers
int_reg_clr            clear some interrupt registers
```

Interrupts such as:

```
en_index       en_posbrk
en_probe
```

are immediately disabled after their first occurrence. The rest remain enforced and can only be disabled by instructions `DISABL_INT` and `DISABL2_INT`.

This page intentionally blank.

8 DSPL Command Set

Reference

DSPL Command Summary

The Mx4 DSPL programming language includes many commands and programming tools. DSPL consists of twelve major command categories. Each category extends the power and flexibility of Mx4 in general areas of motion control.

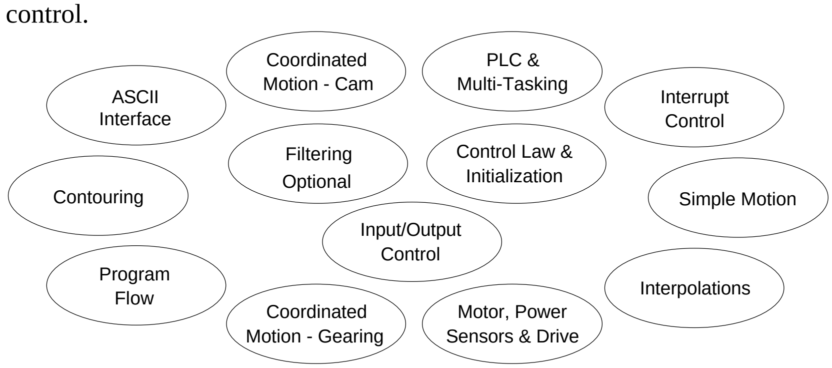


Fig. 8-1: DSPL Command Categories

Control Law & Initialization

Control gains, system parameters, time, position, and velocity units all fall in this category.

COMMAND	DESCRIPTION
CTRL	position, velocity loop control law parameters
CTRL_KA	program an acceleration feed-forward gain
ESTOP_ACC	specify emergency stop maximum acceleration
KILIMIT	integral gain limit
MAXACC	specify maximum acceleration
OFFSET	amplifier offset cancellation
OUTGAIN	position loop output gain
POS_PRESET	preset position counters
POS_SHIFT	position counter reference shift
RESET	reset Mx4 controller card
STEPPER_ON	select stepper / servo axes
SYNC	define Mx4 master/slave status
TRQ_LIMIT	specify a torque limit

Simple Motion

The instructions within this category control the torque, velocity, and position of one or multiple axes with a trapezoidal profile. The commands in this category may be classified as open and closed loop.

COMMAND	DESCRIPTION
AXMOVE	trapezoidal axis move
AXMOVE_S	s-curve axis move
AXMOVE_T	time based axis move
DDAC	direct 18-bit DAC command (open loop)
REL_AXMOVE	relative position axis move
REL_AXMOVE_S	relative s-curve axis move
REL_AXMOVE_T	time based relative axis move
STOP	stops the motion
VELMODE	velocity mode

PLC & Multi-Tasking

The commands in this category start or stop one or several motion control programs (multi-tasking). These commands are used within the PLC program.

COMMAND	DESCRIPTION
`PLC_PROGRAM`	indicates start of PLC program
`RUN_M_PROGRAM`	begin execution of specified program(s)
`STOP_ALL_M_PROGRAM`	stop execution of all running motion programs
`STOP_M_PROGRAM`	stop execution of specified motion program(s)

Input / Output Control

These commands are used to control the status of the Mx4 discrete inputs and outputs.

COMMAND	DESCRIPTION
`INP_STATE`	configure logic state of inputs
`OUTP_OFF`	set status of outputs to low logic level
`OUTP_ON`	set status of outputs to high logic level
`POSBRK_OUT`	set outputs after position breakpoint interrupt

Program Flow Control

The Program Flow Control commands simplify the Mx4 DSPL program flow. Commands within this category include: subroutine call, conditional branching, and other logical instructions. These directives help simplify the development of motion control programs.

COMMAND	DESCRIPTION
`CALL`	initiate execution of subroutine
`DELAY`	halt program execution for specified time
`ELSE`	else operand of if-else-endif structure
`END`	indicates program end
`ENDIF`	endif operand of if-else-endif structure
`IF`	if operand of if-else-endif structure
`RET`	return from subroutine
`WAIT_UNTIL`	halt program execution based on condition
`WAIT_UNTIL_RTC`	halt program execution until signaled by host
`WEND`	wend operand of while-wend structure
`WHILE`	while operand of while-wend structure

Contouring

The Mx4 DSPL includes contouring commands for users who need to generate arbitrary motion profiles. In these applications, a host computer generates position and velocity data points for a complex contouring path in a periodic basis. In CNC and robotics applications, motion trajectories may be computed in real time. These trajectories are transmitted to Mx4 in blocks of position/velocity points. The ring buffer area of Mx4's dual port RAM is the storage area for these motion blocks. Mx4 performs high order interpolation on all these points and executes the trajectory path on a point to point basis.

COMMAND	DESCRIPTION
`BTRATE`	block transfer rate
`CUBIC_INT`	start the internal cubic spline table
`CUBIC_RATE`	set cubic spline point transfer rate
`CUBIC_SCALE`	scales position/velocities, also shifts positions
`START`	start contouring motion
`VECCHG`	contouring vector change

Motor, Power, Sensors and Drive
(available with Mx4/Vx4++, Octavia/Vx8++, and Mx42Turbo only)

Mx4, Octavia, and Mx42Turbo, when equipped properly, can perform all of the signal processing functions of servo amplifier control boards. Control capabilities include commutation, current loops, field current, torque current, current limiting, pulse-width modulation frequency, etc. When properly equipped, these controllers are compatible with all power devices, industrial motors, and a majority of sensors on the market.

COMMAND	DESCRIPTION
`CURR_LIMIT`	current limit setting
`CURR_OFFSET`	current loop offset adjustment
`CURR_PID`	program current loop control law parameters
`ENCOD_MAG`	specify encoder lines, motor poles, comm. option
`FLUX_CURRENT`	bipolar field flux value
`MOTOR_PAR`	set the motor parameter

Motor, Power, Sensors and Drive (cont.)

COMMAND	DESCRIPTION
MOTOR_TECH	define the motor technology
PWM_FREQ	set output PWM signal frequency
Vx4_BLOCK	block further instructions to Vx4++
VIEWVEC	specify Vx4++ parameters to view

Coordinated Motion - Gearing

Multi-axis motion control applications require synchronization of two or more axes in a coordinated task. In addition to the electronic gearing master/slaving technique, compensation tables also help users specify their own application specific "slaving function".

COMMAND	DESCRIPTION
GEAR	unconditional 'electronic' gearing
GEAR_OFF	disengage 'electronic' gearing
GEAR_OFF_ACC	turns electronic gearing off and halt slave(s)
GEAR_POS	'electronic' gearing based on position value
GEAR_PROBE	'electronic' gearing based on external interrupt
REL_AXMOVE_SLAVE	superimposes a relative axis move onto a slave engaged in gearing

Coordinated Motion - Cam

Multi-axis motion control applications require synchronization of two or more axes in a coordinated task. A subset of table oriented master/slaving is known as "electronic cam".

COMMAND	DESCRIPTION
CAM	turns electronic cam on
CAM_OFF	turns only electronic cam off
CAM_OFF_ACC	turns electronic cam off and halts slave(s)
CAM_POINT	place CAM point into CAM table
CAM_POS	turns electronic cam on at a specified position
CAM_PROBE	turns electronic cam on after PROBE is set high

Single & Multi-Dimensional Interpolation

The Mx4 DSPL offers a comprehensive set of linear and circular interpolation commands. All interpolations work on single or multi-dimensional moves. For example, a four-dimensional linear move transfers the system from any arbitrary position, velocity point to another position, velocity point (both defined in multi-dimensional space) with the specified acceleration and jerk. This powerful command yields a well-controlled landing from one trajectory to another.

An example of such a move is rapid acceleration to a position at a specified feed rate and turning to a new trajectory at the same feed rate. It is essential to simultaneously control position, velocity, acceleration, and jerk trajectories in applications like CNC, machine tool, and robotics. The Mx4 circular interpolation command enables several circles to be cut simultaneously. In addition, tables are provided for compensation for reversing error, friction, machine non-linearities, or other forms of inherent mechanical inaccuracies. Cubic splines are computed to interpolate between the intermediate points in a motion segment. This interpolation provides the finest path between any two points with no position, velocity, or acceleration discontinuity at segment boundaries.

COMMAND	DESCRIPTION
CIRCLE	circular interpolation motion
LINEAR_MOVE	constant accel linear motion
LINEAR_MOVE_S	linear, s-curve motion
LINEAR_MOVE_T	linear , simple time-based constant acceleration
OVERRIDE	set feedrate override for LINEAR / CIRCLE
SINE_OFF	disable sine tables for circular interpolation
SINE_ON	enable sine tables for circular interpolation
TABLE_OFF	disable circular interpolation compensation tables
TABLE_ON	enable circular interpolation compensation tables
TABLE_SEL	select a compensation table

Interrupt Control

The Mx4 DSPL includes a comprehensive set of instructions to handle interrupts. There are many system conditions that require the host's and/or DSPL program's immediate attention for an executive (or system-level) decision. Some interrupts will be issued concurrently requiring immediate action by the Mx4. The complete set of interrupts provided by Mx4 facilitates data reporting to the host for issues of system level significance.

COMMAND	DESCRIPTION
DISABL_INT	disable the interrupts
DISABL2_INT	disable the interrupts
EN_BUFBRK	contouring buffer breakpoint interrupt enable
EN_ENCFLT	encoder fault interrupt
EN_ERR	following error interrupt enable
EN_ERRHLT	following error / halt interrupt enable
EN_INDEX	index pulse interrupt enable
EN_MOTCP	motion complete interrupt enable
EN_POSBRK	position breakpoint interrupt enable
EN_PROBE	general purpose external probe interrupt enable
INT_HOST	generate a host interrupt from DSPL program
INT_REG_ALL_CLR	clear all interrupt bit registers
INT_REG_CLR	clear specified interrupts in bit registers

ASCII Interface

COMMAND	DESCRIPTION
INPUT	receive valve from terminal value to terminal
PRINT	send value to terminal
PRINTS	send ASCII string to terminal

Filtering (optional)

COMMAND	DESCRIPTION
LOW_PASS	implement low pass filter at controller output
NOTCH	implement notch filter at controller output

DSPL Command Set

The DSPL command set includes commands, functions, operators, and identifiers listed in alphabetical order. The command listing follows this format:

FUNCTION	indicates the command function
EXECUTION	indicates the amount of time this function will require to execute under worst-case circumstances
SYNTAX	proper command syntax [1]
USAGE	indicates the command usage as follows: Host — host-programming command DSPL — DSPL programming command (PLC) — command may be used in PLC programs (Motion) — command may be used in Motion programs
ARGUMENTS	command arguments (if any) are defined
DESCRIPTION	explanation of command operation, functionality
SEE ALSO	listing of related commands
APPLICATION	some helpful suggestions as to for which applications a command may be useful
EXAMPLE	an example illustrating the command in use

Note: Operators and Identifiers are labeled as such in the listing.

The syntax for many multi-axis commands includes an *n* argument that specifies the command axes and the data arguments for each of the specified axes. For example, the proper syntax for the following error interrupt command is,

```
EN_ERR (n,fer1, ... , fer8)
```

where fer_x is the data argument for axis x. The data arguments follow *n* in a lower to higher axis order. For example, a following error interrupt command involving axes 2 and 4 would appear as,

```
EN_ERR(0xA,fer2,fer4)
```

The *n* argument is a hexadecimal bit coding following the format 0x? where ? is the axis mask,

axis mask	bit 0	axis 1
	bit 1	axis 2
	bit 2	axis 3
	bit 3	axis 4
	bit 4	axis 5
	bit 5	axis 6
	bit 6	axis 7
	bit 7	axis 8

For example, 0x3 bit codes axes 1 and 2; 0xE bit codes axes 2, 3, 4, etc.

ABS

FUNCTION Calculate the Absolute Value of a Constant or a Variable Value.

EXECUTION 10 microseconds

SYNTAX `ABS(valu)` *or* `-ABS(valu)`

USAGE DSPL (PLC, Motion)

ARGUMENTS

valu — A constant real number
or a
variable (`VAR1` through `VAR128`)

DESCRIPTION

This mathematical function calculates the absolute value of a constant or a variable value. If a minus sign appears to the left of the ABS function, the result of the absolute value calculation is multiplied by -1.

Note: This function can only be used with a variable assignment statement. For example:

```
VAR55 = ABS(VAR32)
```

SEE ALSO `FRAC, INT, SIGN, SQRT`

EXAMPLE

The first example calculates the absolute value of the value stored in `VAR36` and stores the negated result in `VAR49`:

```
VAR49 = -ABS(VAR36)
```

The second example finds the absolute value of -6.751 and stores the result (6.751) in `VAR51`:

```
VAR51 = ABS(-6.751)
```

ADC1, ADC2, ADC3, ADC4 — IDENTIFIER

IDENTIFIER Analog-to-digital input values.

USAGE DSPL (PLC, Motion)

DESCRIPTION

If the Mx4 controller includes the Mx4 Quad ADC Acc4 option, four (4) analog-to-digital (ADC) values are available in DSPL programs. The value (in Volts) that is stored in each of the ADC values corresponds to the voltage applied to the ADC input.

Name	Description
`ADC1`	analog input 1
`ADC2`	analog input 2
`ADC3`	analog input 3
`ADC4`	analog input 4

SEE ALSO none

EXAMPLE

The ADC values can be used as follows:

- To assign the value of a variable:

  ```
  VAR23 = ADC3
  ```

  sets `VAR23` to the value (in Volts) of the analog-to-digital input 3 voltage. For instance, applying -1.25 volts across the `ADC3` input would result in `VAR23` being set to -1.25.

- as one of the values used in conjunction with a DSPL arithmetic operation:

  ```
  VAR12 = ADC2 - 1.5
  ```

- as one of the arguments in a DSPL conditional expression:

  ```
  WHILE(ADC4 <= VAR33)
  ```

AND, OR

OPERATOR

OPERATOR Logical AND, Logical OR

SYNTAX

```
(expression1) AND (expression2)
(expression1) OR (expression2)
```

USAGE DSPL (PLC, Motion)

ARGUMENTS

expression1 A DSPL conditional expression

expression2 A DSPL conditional expression

DESCRIPTION

This operator performs the logical AND or the logical OR of two DSPL conditional expressions. For the operator `AND`, the result is TRUE (1) only if both of the conditional expressions evaluated as TRUE, otherwise the result is FALSE (0). For the operator `OR`, the result is FALSE (0), only if both of the conditional expressions evaluated as FALSE (0), otherwise the result is TRUE (1).

Note: These operators can only be used in a DSPL conditional statement inside of a DSPL conditional structure (i.e. `IF`, `WHILE`, or `WAIT_UNTIL`). For example:

```
WHILE((INP1_REG & 0x09) AND (INP2_REG & 0x02))
```

SEE ALSO `~, &, IF, WHILE, WAIT_UNTIL`

AND, OR cont.

OPERATOR

EXAMPLE

The WAIT_UNTIL statement below will stop the execution of the DSPL code as long as both of the following are true: the actual position of axis 1 is less than 1000, and the actual velocity of axis 2 is greater than the value stored in VAR29:

```
WAIT_UNTIL((POS1 < 1000) AND (VEL2 > VAR29))
```

The next WAIT_UNTIL statement will stop the execution of the DSPL code as long as either of the following is true: the actual velocity of axis 2 is less than or equal to 2.5, or the actual position of axis 2 is greater than the value stored in VAR9:

```
WAIT_UNTIL((CVEL1 <= 2.5) OR (POS2 > VAR9))
```

ARCTAN

FUNCTION Calculate the Arctangent of a Constant or a Variable Value.

EXECUTION 50 microseconds

SYNTAX `ARCTAN(valu)` *or* `-ARCTAN(valu)`

USAGE DSPL (PLC, Motion)

ARGUMENTS

valu A constant real number
or a
variable (VAR1 through VAR128)

DESCRIPTION

This mathematical function calculates the arctangent of a constant or a variable value. The result will be in the range $-\pi/2$ to $\pi/2$. If *valu* is a constant and a minus sign appears to the left of the ARCTAN function, the result of the arctangent calculation is multiplied by -1.

Note: This function can only be used with a variable assignment statement. For example:

```
VAR62 = -ARCTAN(83.33)
```

SEE ALSO COS, SIN, TAN

EXAMPLE

The first example calculates the arctangent of the value stored in VAR5 and stores the result in VAR14:

```
VAR14 = ARCTAN(VAR5)
```

The second example finds the arctangent of -49.63 and stores the result (-1.55064995) in VAR31:

```
VAR31 = ARCTAN(-49.63)
```

AXMOVE

FUNCTION Axis Move with Trapezoidal Trajectory

EXECUTION 200 microseconds

SYNTAX `AXMOVE (n, acc1, pos1, vel1, ... , acc8, pos8, vel8)`

USAGE DSPL (Motion), Host (command code: 60h)

ARGUMENTS

n — bit coding of the specified axis(es)

acc_x — unsigned value specifying the maximum halting acceleration (deceleration) for axis x

$0 \le acc_x \le 1.999969$ counts/(200µs)^2

pos_x — target position for axis x

$-2147483648 \le pos_x \le 2147483647$ counts

vel_x — unsigned target velocity for axis x

$0 \le vel_x \le 255.99998$ counts/200µs

When used in DSPL, arguments acc_x, pos_x and vel_x may be selected as variables.

DESCRIPTION

The `AXMOVE` command allows for trapezoidal command generation with specified endpoint position, slew rate velocity, and acceleration for each axis. This command is suitable for linear moves.

SEE ALSO `AXMOVE_S, AXMOVE_T, REL_AXMOVE, REL_AXMOVE_S, REL_AXMOVE_T, STOP`

AXMOVE cont.

APPLICATION

This command can be used in almost any imaginable motion control application. Applications may benefit from this command any time there is a need for a linear move from point A to point B in a multi-dimensional space. To name a few applications: pick and place robots (e.g., in component insertion), rapid traverse (e.g., in machining), and master/slaving (e.g., in paper processing and packaging) applications.

Command Sequence Example

```
MAXACC ( )      ;set the maximum accel. so system can be stopped
CTRL ( )        ;set the gain values
KILIMIT ( )
AXMOVE ( )      ;run system in axis move (linear trapezoidal) mode
:
EN_MOTCP ( )    ;enable motion complete
                ;upon the completion of this (command) trajectory
                ;Mx4 generates motion complete interrupt
```

EXAMPLE 1

Assuming current positions of zero for axes 1 and 2, we want to move axis 1 to the target position of 234567 and axis 2 to the target position of -3000 counts. Let's also assume that we want this move to be accomplished with the slew rate velocity of 4.0 counts/200µs for axis 1 and 3.50 counts/200µs for axis 2, and an acceleration of 0.005 counts/(200 µs)^2 for both axes.

```
AXMOVE (0x3,0.005,234567.0,0.005,-3000,3.50)
```

EXAMPLE 2

The user can issue a new axis move command before the motion of the previous `AXMOVE` command is completed. For example, assume the `AXMOVE` command of Example 1 is executed. Now, the DSPL Motion program 'decides' to stop axis two at a new target position of -50000 counts with a new slew rate of 8.0 counts/200µs and a new acceleration of 0.035 counts/(200µs)^2. While the `AXMOVE` of Example 1 is in progress, the DSPL Motion program issues the new command.

```
AXMOVE (0x2,0.035,-50000,8.0)
```

AXMOVE_S

FUNCTION S-Curve Axis Move with Trapezoidal Trajectory

EXECUTION 200 microseconds

SYNTAX `AXMOVE_S (n, acc1, pos1, vel1, ... , acc8, pos8, vel8)`

USAGE DSPL (Motion), Host (command code: 82h)

ARGUMENTS

n — bit coding of the specified axis(es)

acc_x — unsigned value specifying the acceleration/deceleration for axis x

$0 \le acc_x \le 1.999969$ counts/(200µs)^2

pos_x — target position for axis x

$-2147483648 \le pos_x \le 2147483647$ counts

vel_x — unsigned target velocity for axis x

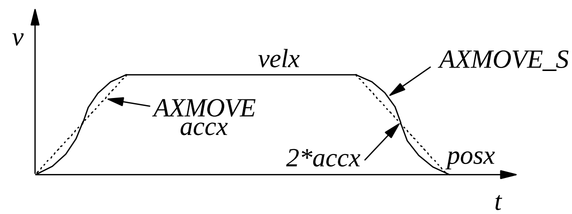
$0 \le vel_x \le 255.99998$ counts/200µs

When used in DSPL, arguments acc_x, pos_x, and vel_x may be selected as variables.

DESCRIPTION

The `AXMOVE_S` command allows for s-curve command generation with specified endpoint position, slew rate velocity, and acceleration for each axis. This command is suitable for linear moves where s-curve acceleration is desired.

AXMOVE_S cont.



The figure above illustrates the velocity profile of the `AXMOVE_S` along with the linear velocity ramp of the `AXMOVE` command. With `AXMOVE_S,` the acceleration will reach a value of 2*accx for a maximum (see above figure).

SEE ALSO `AXMOVE, AXMOVE_T, REL_AXMOVE, REL_AXMOVE_S, REL_AXMOVE_T, STOP`

APPLICATION

Refer to *DSPL Application Programs*.

EXAMPLE 1

Assuming current positions of zero for axes 1 and 2, we want to move axis 1 to the target position of 200000 counts and axis 2 to the target position of -3000 counts. Let's also assume that we want this move to be accomplished with the slew rate velocity of 4.0 counts/200 µs for axis 1 and 2.0 counts/200 µs for axis 2. Use an acceleration reference of 0.05 counts/(200 µs)^2 for both axes.

```
AXMOVE_S (0x3, .05, 200000, 4.0, .05, -3000, 2.0)
```

AXMOVE_T

FUNCTION Time-Based Axis Move with Trapezoidal Trajectory

EXECUTION 200 microseconds

SYNTAX `AXMOVE_T (n, acc`$_1$`, pos`$_1$`, tm`$_1$`, ... , acc`$_8$`, pos`$_8$`, tm`$_8$`)`

USAGE DSPL (Motion), Host (command code: 8Fh)

ARGUMENTS

n — bit coding of the specified axis(es)

acc_x — unsigned value specifying the acceleration/deceleration for axis x

$0 \leq acc_x \leq 1.999969$ counts/(200µs)^2

pos_x — target position for axis x

$-2147483648 \leq pos_x \leq 2147483647$ counts

tm_x — motion time for axis x

$0 \leq tm_x \leq 5000000$ (200µs)

When used in DSPL, arguments acc_x, pos_x, and tm_x may be selected as variables.

Note: The time argument, tm_x, is an unsigned value with a unit of 200µsec.

DESCRIPTION

The `AXMOVE_T` commands allow for trapezoidal command generation with specified endpoint position, acceleration, and time to complete the move for each axis. This command is suitable for linear moves where endpoint position and motion time are the specifying parameters.

AXMOVE_T cont.

The `AXMOVE_T` command is similar to `AXMOVE`, with the exception that the velocity argument is replaced with a time argument. `AXMOVE_T` will automatically calculate a suitable slew rate velocity to achieve the programmed endpoint position in the programmed amount of time, following a trapezoidal velocity profile (similar to `AXMOVE`).

SEE ALSO `AXMOVE, AXMOVE_S, REL_AXMOVE, REL_AXMOVE_S, REL_AXMOVE_T, STOP`

APPLICATION

Refer to *DSPL Application Programs*.

EXAMPLE

Move axis 1 to the target position of 10000 counts and axis 3 to the target position of 3599 counts. Let's assume that we want this move to be accomplished with the acceleration reference of 0.56 counts/(200 $\mu s)^2$ and a time of 50msec (250*200µsec) for both axes.

```
AXMOVE_T (0x5, .56, 10000, 250, .56, 3599, 250)
```

BTRATE

FUNCTION Set 2nd Order Contour Block Transfer Rate

EXECUTION 10 microseconds

SYNTAX `BTRATE (m)`

USAGE DSPL (Motion), Host (command code: 73h)

ARGUMENTS

m — selects the block transfer rate for all of the axes.
m is an integer ranged from 0 to 3

m=0	block transfer rate is 5 ms per point
m=1	block transfer rate is 10 ms per point
m=2	block transfer rate is 15 ms per point
m=3	block transfer rate is 20 ms per point

DESCRIPTION

This command sets the 2nd order contouring block transfer rate for the system. For example, if the block transfer rate is set at 10 ms, the time interval between each point in the ring buffer is '10 ms' (e.g., the DSP will interpolate each point for 10 ms).

Note 1: The host should not adjust the block transfer rate when contouring is in process.

Note 2: The default block transfer rate is set at 5 ms per point.

SEE ALSO `CUBIC_RATE`

BTRATE cont.

APPLICATION

This command is useful in 2nd order contouring applications. Depending on the capability of the host processor, position/velocity points on multi-dimensional trajectories may be broken down to the points that (timewise) may be near or far from each other. Clearly, slower CPUs are capable of breaking down geometries to position and velocity points that are widely spaced in time. This instruction makes the time interval in between the two adjacent points (in contouring) programmable. Please remember that regardless of the value programmed for this time interval (5, 10, 15 or 20 ms), Mx4 will internally perform a high-order interpolation of the points breaking them down to 200 μs.

Command Sequence Example

See `EN_BUFBRK`

EXAMPLE

Set a contouring interpolation interval of 10 ms.

```
BTRATE (1)
```

CALL

FUNCTION Subroutine Calls

EXECUTION 10 microseconds

SYNTAX `CALL (program label)`

USAGE DSPL (Motion)

ARGUMENTS

program label the name of the subroutine to be called

DESCRIPTION

This instruction is used to call a subroutine from a Motion program. Program flow after a `CALL` instruction continues at the start of the subroutine called. Program flow returns to the calling Motion program after the RET instruction.

SEE ALSO `RET`

EXAMPLE

Call the subroutine "HALT_AX1".

```
CALL (HALT_AX1)
```

CAM

FUNCTION Engage Electronic Cam

EXECUTION 200 microseconds

SYNTAX `CAM (n, m, tablestart1, tablesize1 ... , tablestart8, tablesize8)`

USAGE DSPL (Motion), Host (command code: A4h)

ARGUMENTS

n	bit coding the ONLY master axis
m	bit coding the slave axis(es)
$tablestart_x$	specifies cam table start index for slave axis x $0 <= tablestart_x <= 1600$
$tablesize_x$	specifies cam table size for slave axis x $3 <= tablesize_x <= 1600$

When used in DSPL, arguments tablestart and tablesize may be either constants or DSPL variables.

DESCRIPTION

The commands making up the electronic cam feature are; `CAM`, `CAM_OFF`, `CAM_OFF_ACC`, `CAM_POINT`, `CAM_POS`, and `CAM_PROBE`. DSPL keywords `[CAMCOUNT1-8, Mx4 Octavia] [CAMCOUNT1-4, Mx4] [CAMCOUNT1-2, Mx42]`.

The Mx4 controller is capable of storing up to 1600 cam points. Each cam point consists of a master relative position, and an associated slave relative position. A cam table can be between 3 and 1600 cam points long, and the user may define any number of cam tables in the 1600-point cam table capacity. Cam commands utilize tablestart and tablesize arguments to specify which 'portion' of the 1600-point cam table region to 'run' on.

Cam table points may be downloaded in file format from within Mx4pro or built from within DSPL using the `CAM_POINT` command. The `CAM_POINT` command may also be used to modify cam points 'on the fly'. The

CAM cont.

DSPL identifiers `CAMCOUNT1,2,3,etc.` indicate at which cam table indices the slave axes(es) are ‘at’ (`CAMCOUNT1` is for axis 1, etc.).

The cam points consist of relative position values for master and slave. The first cam point in a table must be 0, 0. The last point in a cam table is the cycle length for master and slave. For example, if the full cam cycle for a master axis is 5000 counts and the slave would travel -1024 counts in that cycle, the last cam point in that cam table would be 5000, -1024. Note that the master/slave position ratios can not exceed the range [-256 to 255,999]. Also, the minimum ratio is +/- 1/128. For example, for 1000 counts of the master axis, the slave axis(es) can not have more than -256000 counts in the negative direction or 255999 counts in the positive direction.

The slave axes utilize the `MAXACC` acceleration value as the maximum acceleration the slave axes can reach while following the electronic cam trajectory, and therefore must be programmed before cam operation. This command turns on the mechanical cam function for the selected master and slave(s). The slave(s) follow the master according to the master/slave position pairs stored in the cam table. The slave axis(es) utilize `MAXACC` as the maximum acceleration they can achieve in following the master trajectory.

Note: Activation of *ESTOP during cam operation will halt the master axis, and subsequently the slave axis(es). Slave(s) remain “engaged” in cam mode after the input-triggered halt.

SEE ALSO `CAM_OFF, CAM_OFF_ACC, CAM_POINT, CAM_POS, CAM_PROBE, MAXACC, SYNC`

APPLICATION

General master/slaving, in particular packaging, synchronous cutting, flying shear, and mark registration, require the coordination of several axes in cam fashion. For these applications, the user is required to load the cam function along with the position spacing that defines the distance between the adjacent gear ratios stored in the cam table.

CAM cont.

EXAMPLE

Set axis 1 as the master axis, axes 2 and 3 as slaves. The axis 2 slave will use the 10-point cam table beginning at index 0, while the axis 3 slave will use the 25 point cam table beginning at index 100.

```
CAM(0x1,0x6,0,10,100,25)
```

CAM_OFF

FUNCTION Turns Off, Disengages Cam Slave Axis(es)

EXECUTION 10 microseconds

SYNTAX `CAM_OFF (n)`

USAGE DSPL (Motion), Host (command code: A7h)

ARGUMENTS

n bit coding the slave axis(es) to be disengaged

DESCRIPTION

This command disengages the system that was under master slave control.

SEE ALSO `CAM, CAM_OFF_ACC, CAM_POINT, CAM_POS, CAM_PROBE, SYNC`

APPLICATION

General master/slaving, in particular packaging, synchronous cutting, flying shear, and mark registration, require the coordination of several axes in cam fashion. For these applications, the user is required to load the cam function along with the position spacing that defines the distance between the adjacent gear ratios stored in the cam table.

EXAMPLE

Immediately disengage slave axes 3 and 4 from the master axis.

```
CAM_OFF(0xc)
```

CAM_OFF_ACC

FUNCTION Turns Off, Disengages Cam Slave Axis(es) With Acceleration

EXECUTION 50 microseconds

SYNTAX `CAM_OFF_ACC (n)`

RTC CODE DSPL (Motion), Host (command code: A8h)

ARGUMENTS

n bit coding the slave axis(es) to be disengaged

DESCRIPTION

This command disengages the system that was under master slave control. The slave axis(es) will come to a stop at the maximum acceleration rate programmed by `MAXACC`.

SEE ALSO `CAM`, `CAM_OFF`, `CAM_POINT`, `CAM_POS`, `CAM_PROBE`, `SYNC`

APPLICATION

General master/slaving, in particular packaging, synchronous cutting, flying shear, and mark registration, require the coordination of several axes in cam fashion. For these applications, the user is required to load the cam function along with the position spacing that defines the distance between the adjacent gear ratios stored in the cam table.

EXAMPLE

Disengage with acceleration profile slave axes 3 and 4 from the master axis.

```
CAM_OFF_ACC(0xc)
```

CAM_POINT

FUNCTION Place Cam Point Into Cam Table

EXECUTION 200 microseconds

SYNTAX `CAM_POINT (tablestart, tablesize, index, masterpos, slavepos)`

USAGE DSPL (Motion), Host (command code: B3h)

ARGUMENTS

tablestart — specifies cam table start index

0 <= tablestart <= 1600

tablesize — specifies cam table size

3 <= tablesize <= 1600

index — specifies index at which to place the cam point

0 <= index <= (tablesize-1)

masterpos — cam point master axis relative position
slavepos — cam point slave axis relative position

When used in DSPL, arguments tablestart, tablesize, index, masterpos, and slavepos may be either constants or DSPL variables.

DESCRIPTION

The `CAM_POINT` allows the user to either build entire cam tables from within the DSPL environment or alternatively, edit cam table points (i.e.: change cam points ‘on the fly’). Cam table points consist of master, slave position pairs, and cam tables can be anywhere from 3 to 1600 cam points long. The first point of a cam table (index = 0) must be 0,0. The last point of a cam table (index = tablesize-1) is mastercyclelength, slavecyclelength; where the cycle lengths for the master and slave are the relative cam cycle lengths (i.e.: master cycle length is 4096 counts, the slave cycle length is 1024 counts, for a full cycle ratio of 4:1). Cam master/slave position ratios can not exceed the range [-256 to 255,999]. Also, the minimum ratio is +/- 1/128.

CAM_POINT cont.

SEE ALSO `CAM, CAM_OFF, CAM_OFF_ACC, CAM_POS, CAM_PROBE, SYNC`

APPLICATION

See Application Notes.

EXAMPLE

A 10-point cam table exists at table start index 500. Replace the 3rd point of the table with the master, slave point 1000, 3000.

```
CAM_POINT (500, 10, 2, 1000, 3000)
```

CAM_POS

FUNCTION Turns Electronic Cam On at a Specified Position

EXECUTION 200 microseconds

SYNTAX CAM_POS (n, m, masterpos$_1$, tablestart$_1$, tablesize$_1$,... , pos$_8$, tablestart$_8$, tablesize$_8$)

USAGE DSPL (Motion), Host (command code: A5h)

ARGUMENTS

n	bit coding the ONLY master axis
m	bit coding the slave axis(es)
masterpos$_x$	specifying the master position value for slave axis x that the electronics cam engages
tablestart$_x$	specifies cam table start index for slave axis x 0 <= tablestart$_x$ <= 1600
tablesize$_x$	specifies cam table size for slave axis x 3 <= tablesize$_x$ <= 1600

When used in DSPL, arguments masterpos, tablestart and tablesize may be either constants or DSPL variables.

DESCRIPTION

This command engages at the specified master position the mechanical cam function for the selected master and slave(s). The slave(s) follow the master according to the master/slave position pairs stored in the cam table. The slave axis(es) utilizes MAXACC as the maximum acceleration they can achieve in following the master trajectory.

Note: Activation of *ESTOP during cam operation will halt the master axis, and subsequently the slave axis(es). Slave(s) remain “engaged” in cam mode after the input-triggered halt.

CAM_POS cont.

SEE ALSO `CAM, CAM_OFF, CAM_OFF_ACC, CAM_POINT, CAM_PROBE, SYNC`

APPLICATION

General master/slaving, in particular packaging, synchronous cutting, flying shear, and mark registration, require the coordination of several axes in cam fashion. For these applications, the user is required to load the cam function along with the position spacing that defines the distance between the adjacent gear ratios stored in the cam table.

EXAMPLE

Set axis 4 as the master axis, axes 2 and 3 as slaves. The axis 2 slave will use the 10-point cam table beginning at index 0, while the axis 3 slave will use the 25-point cam table beginning at index specified in VAR8. Axis 2 slave should engage when the master axis is at position 1000, and axis 3 slave should engage when the master axis is at position 4096.

```
CAM_POS(0x8,0x6,1000,0,10,4096,VAR8,25)
```

CAM_PROBE

FUNCTION Turns Electronic Cam On After Probe Input

EXECUTION 200 microseconds

SYNTAX `CAM_PROBE (n, m, q, tablestart`$_1$`, tablesize`$_1$` ... , tablestart`$_8$`, tablesize`$_8$`)`

USAGE DSPL (Motion), Host (command code: A6h)

ARGUMENTS

n	bit coding the ONLY master axis
m	bit coding the slave axis(es)
q	specifies the *EXTx probe interrupt to be used

[Mx4]

q=1	:	*EXT1
q=2	:	*EXT2

[Mx4 Octavia]

q=1	:	*EXT1
q=2	:	*EXT2
q=4	:	*EXT3
q=8	:	*EXT4

$tablestart_x$ specifies cam table start index for slave axis x

$0 <= tablestart_x <= 1600$

$tablesize_x$ specifies cam table size for slave axis x

$3 <= tablesize_x <= 1600$

When used in DSPL, arguments tablestart and tablesize may be either constants or DSPL variables.

CAM_PROBE cont.

DESCRIPTION

This command engages at the occurrence of the specified external interrupt (*EXT1,2,3,4) the mechanical cam function for the selected master and slave(s). The slave(s) follow the master according to the master/slave position pairs stored in the cam table. The slave axis(es) utilizes MAXACC as the maximum acceleration they can achieve in following the master trajectory.

Note: Execution of the CAM_PROBE command will disable any previously enabled EN_PROBE interrupt. Probe input (*EXT1, *EXT2, *EXT3, or *EXT4) activation does not generate an interrupt with the CAM_PROBE command.

Note: Activation of *ESTOP during cam operation will halt the master axis, and subsequently the slave axis(es). Slave(s) remain “engaged” in cam mode after the input-triggered halt.

SEE ALSO

CAM, CAM_OFF, CAM_OFF_ACC, CAM_POINT, CAM_POS, SYNC

APPLICATION

General master/slaving, in particular packaging, synchronous cutting, flying shear, and mark registration, require the coordination of several axes in cam fashion. For these applications, the user is required to load the cam function along with the position spacing that defines the distance between the adjacent gear ratios stored in the cam table.

EXAMPLE

Set axis 2 as the master axis, set axes 1 and 3 as slaves. The axis 1 slave will use the 100-point cam table beginning at index 0, while the axis 3 slave will use the 250-point cam table beginning at index specified in VAR38. Engage slave axes in cam at occurrence of *EXT2 interrupt.

```
CAM(0x2,0x5,0x2,0,100,VAR38,250)
```

CAMCOUNT1, …, CAMCOUNT8 IDENTIFIER

IDENTIFIER Slave Axis Table Index Counter

USAGE DSPL (PLC, Motion)

DESCRIPTION

When engaged in CAM motion, the slave axis (es) derive their position with respect to the master position from the master/slave position points which make up the CAM table. The `CAMCOUNTx` identifiers indicate at which CAM point the respective slave axis (es) is located within the CAM table.

SEE ALSO

`CAM, CAM_POINT, CAM_POS, CAM_PROBE, CAM_OFF, CAM_OFF_ACC`

EXAMPLE

Delay DSPL program flow until the axis2 slave axis passes index 19 of the CAM table.

```
WAIT_UNTIL (CAMCOUNT2 > 19)
```

CIRCLE

FUNCTION Circular Trajectory Motion

EXECUTION Depends on size and feedrate of circle

SYNTAX `CIRCLE (n, cent`$_x$`, cent`$_y$`, radius, feedrate, target`$_x$`, target`$_y$`)`

USAGE DSPL (Motion)

ARGUMENTS

n	bit coding the two axes in circular motion
$cent_x$	the circle center's x-axis position component relative to the current x-axis command position $-536870912 <= cent_x <= 536870912$ counts
$cent_y$	the circle center's y axis position component relative to the current y-axis command position $-536870912 <= cent_y <= 536870912$ counts
radius	positive value specifying circle radius radius <= 536870912 counts
feedrate	circle feedrate (velocity), may be positive or negative -256 <= feedrate <= 255.99998 counts/200µs

Note: circle period must be >2 seconds

$target_x$	relative (x-axis component) distance of target from the current x-axis command position $-1073741824 <= target_x <= 1073741824$ counts

CIRCLE cont.

target_y relative (y-axis component) distance of target from the current y-axis command position

-1073741824 <= target_y <= 1073741824 counts

When used in DSPL, arguments, cent_x, cent_y, radius, feedrate, target_x, and target_y may be either constants or DSPL variables.

DESCRIPTION

`CIRCLE` allows the user to program circular motion for two axes. In order to perform the circular interpolation, the user has the option of choosing which interpolation tables are used for the generation of the command position and command velocity. The choices are:

1. Standard sine tables only
2. Sine tables plus user-defined position and velocity compensation tables
3. User-defined position and velocity compensation tables only

The user-defined compensation tables allow the individual user to compensate for both position and velocity non-linearities of the particular system's mechanical parts.

Note: By selecting to use only the user-defined compensation tables, the users may define their own interpolation scheme based on the position and velocity compensation tables.

The command position and velocity profiles are illustrated in Figs. 8-2 and 8-3 for the standard sine table case. Fig. 8-2 depicts the profiles for a positive feedrate while Fig. 8-3 illustrates the profiles for a negative feedrate. It is important to note that with the addition of the compensation tables, the position and velocity profiles of the following figures would be altered.

CIRCLE cont.

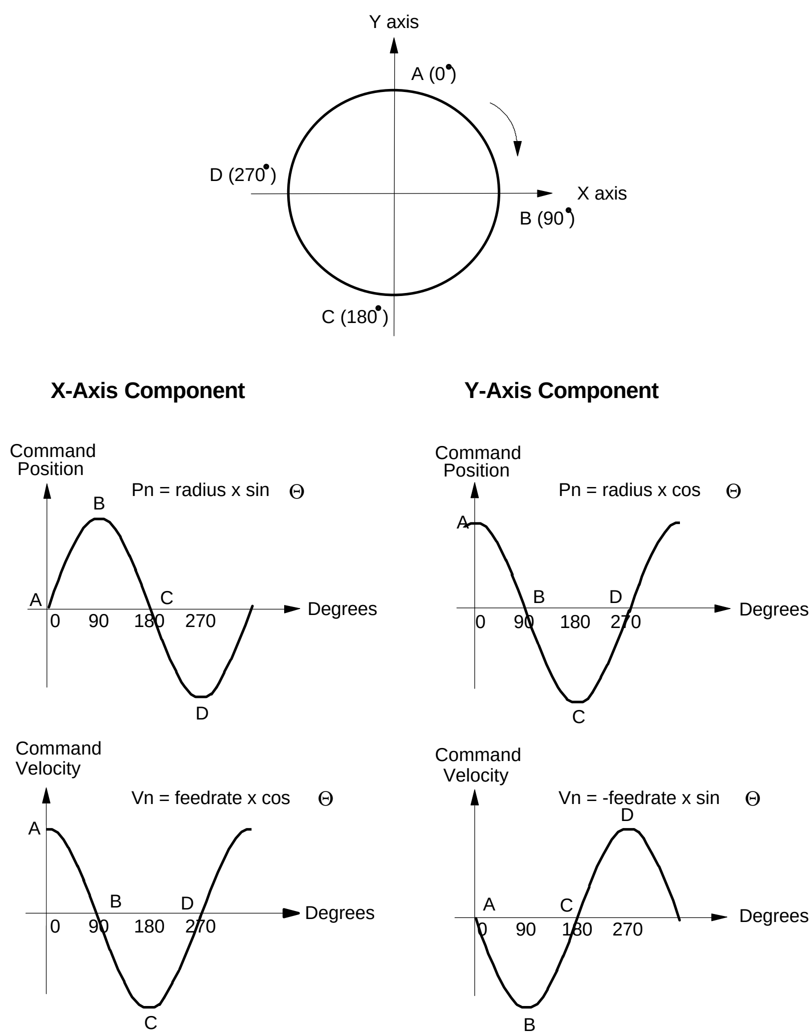


Fig. 8-2: Profiles for Positive Feedrate

CIRCLE cont.

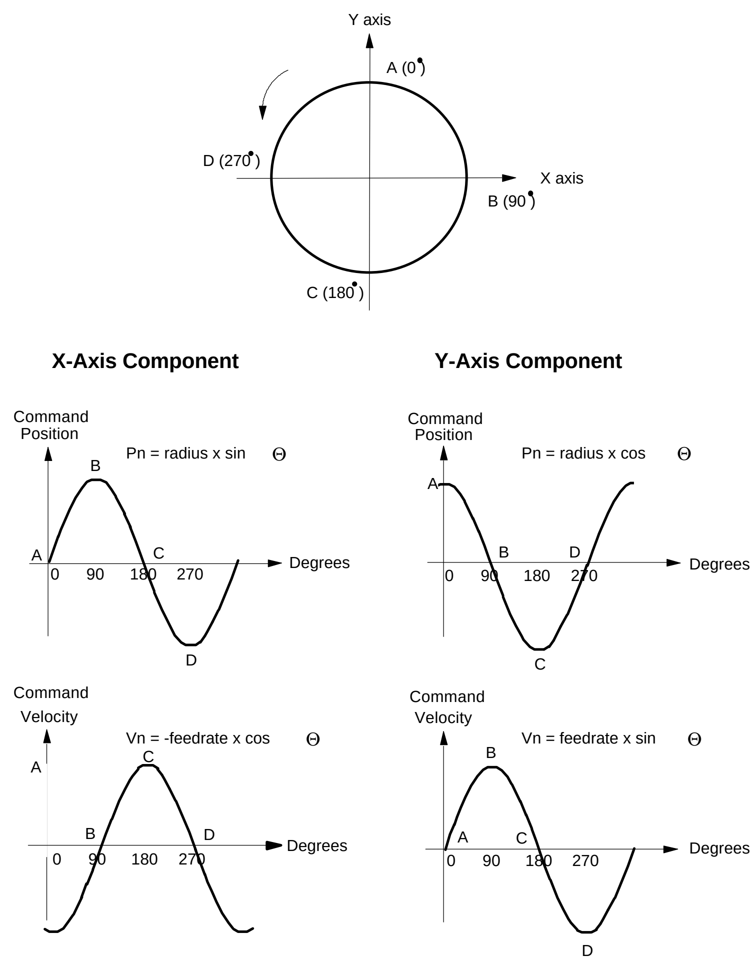


Fig. 8-3: Profiles for a Negative Feedrate

CIRCLE cont.

Upon execution of a `CIRCLE` or `LINEAR` related command, the DSPL program flow will proceed to the following command. If the following command is not a `CIRCLE` or `LINEAR` related command, it will be executed immediately. If the following command is a `CIRCLE` or `LINEAR` related command, it will be executed after the previous `CIRCLE`/`LINEAR` motion is complete.

SEE ALSO `CLEAR_POS_TABLE`, `CLEAR_VEL_TABLE`, `LINEAR_MOVE_`, `LINEAR_MOVE_S`, `LINEAR_MOVE_T`, `LOAD_POS_TABLE`, `LOAD_VEL_TABLE` (*Mx4 User's Guide*), `SINE_OFF`, `SINE_ON`, `TABLE_OFF`, `TABLE_ON`

APPLICATION

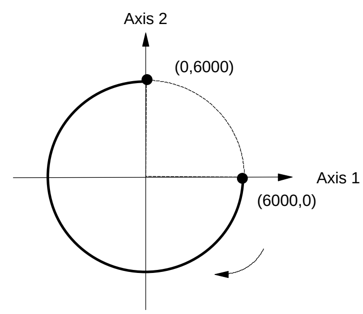
See *Application Notes*

EXAMPLE

Move (axis one, axis two) from a current position of (6000, 0) to a final position of (0, 6000) using circular interpolation with a feedrate equal to 1.5 counts/200µs. The radius of the circle is 6000 counts. Assume standard sine table interpolation.

Note: The axis two velocity must be -1.5 counts/200µs at the starting point of the circle (see velocity profiles as illustrated in Fig. 5-2 and 5-3).

CIRCLE cont.



n	0x3
$cent_x$	-6000 counts
$cent_y$	0 counts
radius	6000 counts
feedrate	1.5 counts/200µs
$target_x$	-6000 counts
$target_y$	6000 counts

```
TABLE_OFF (0x3)       ; Disable compensation tables
SINE_ON (0x3)         ; Enable standard sine tables
CIRCLE (0x3, -6000, 0, 6000, 1.5, -6000, 6000)
```

COS

FUNCTION Calculate the Cosine of a Constant or a Variable Value.

EXECUTION 75 microseconds

SYNTAX `COS(valu)` *or* `-COS(valu)`

USAGE DSPL (PLC, Motion)

ARGUMENTS

valu A constant
or a
variable (`VAR1` through `VAR128`)

DESCRIPTION

This mathematical function calculates the cosine of a constant or a variable value specified in radians. If *valu* is a constant and a minus sign appears to the left of the `COS` function, the result of the cosine calculation is multiplied by -1.

Note: This function can only be used with a variable assignment statement. For example:

```
VAR19 = COS(4.963)
```

SEE ALSO `ARCTAN, SIN, TAN`

EXAMPLE

The first example calculates the cosine of the value stored in `VAR23` and stores the result in `VAR42`:

```
VAR42 = COS(VAR23)
```

The second example finds the cosine of -0.529 radians and stores the negated result (-0.863312172) in `VAR8`:

```
VAR8 = -COS(-0.529)
```

CPOS1, …, CPOS8 — IDENTIFIER

IDENTIFIER Command Position State Variable

USAGE DSPL (PLC, Motion)

DESCRIPTION

A command position state variable holds a 32-bit two's complement integer value that represents the position (in encoder edge counts) that DSPL is commanding the specified axis to reach.

Name	Description
`CPOS1`	axis 1 command position
`CPOS2`	axis 2 command position
`CPOS3`	axis 3 command position
.	
.	
`CPOS8`	axis 8 command position

SEE ALSO `ERR1, INDEX_POS1, POS1, PROBE_POS1, etc.`

EXAMPLE

The command position state variables can be used as follows:

- as one of the values used in conjunction with a DSPL arithmetic operation:

```
VAR12 = CPOS3 + 33000
```

- as one of the arguments in a DSPL conditional expression:

```
WAIT_UNTIL(CPOS1 > 100000)
```

CTRL

FUNCTION Control Law Parameters

EXECUTION 200 microseconds

SYNTAX `CTRL (n, par`$_{11}$`, ... , par`$_{14}$`, ... , par`$_{81}$`, ... , par`$_{84}$`)`

USAGE DSPL (Motion), Host (command code: 62h)

ARGUMENTS

n	bit coding of the specified axis(es)
par_{x1}	unsigned value for Ki gain
par_{x2}	unsigned value for Kp gain
par_{x3}	unsigned value for Kf gain
par_{x4}	unsigned value for Kd gain

$0 \leq par_{xy} \leq 32767$

When used in DSPL, arguments par_{x1}, par_{x2}, par_{x3} and par_{x4} may be selected as variables.

DESCRIPTION

This command performs a state feedback control algorithm combined with a modified PID. The state feedback control algorithm includes an observer which estimates the instantaneous values for speed and acceleration. The feedback loops are then individually commanded to provide a robust control, which is smooth and stable over a wide range of servo operation. In addition, this algorithm performs a modified PID with the saturation threshold set for integral action. A common PID includes two zeros and one pole, which may not be suitable for systems with noisy feedback. Also, the integral part of a common PID algorithm may saturate the registers creating overshoots or other forms of instability. A modified PID includes a second pole to solve the latter problem and a programmable integral limit to solve the former one.

In the modified PID algorithm; par_1, par_2, par_3, and par_4 are values representing the integral, proportional, velocity state feed forward, and differential gains, respectively.

CTRL cont.

Scaling Factors

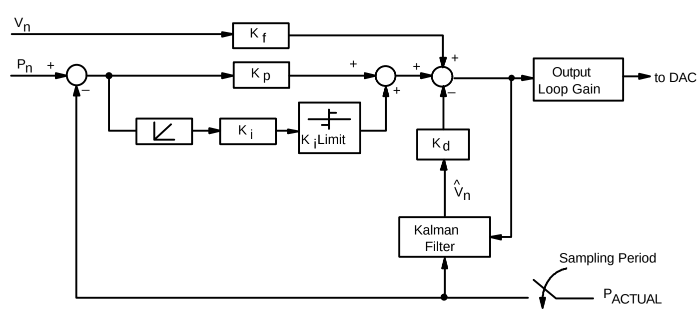
The DSP uses an internal scaling factor for each gain. These factors have been optimally selected for worst case numerical conditions. These factors are:

GAIN	SCALING FACTOR	VALUE
K_f	1.525E-08	v/(c/s)
K_p	0.595E-06	v/c
K_i	3.308E-05	(v/s)/c
K_d	1.9875E-08	v/(c/s)
Output Loop Gain	integer	NA
v = volts, c = encoder edge counts, s = seconds		

For example,

100 counts of position error and K_p of 1000 (other gains are zero) will result in an output voltage of 59.5 millivolts.

i.e. $100 \times 1000 \times 0.595E\text{-}06 = 59.5$



Block Diagram of Control Law

SEE ALSO `KILIMIT, OFFSET, OUTGAIN`

CTRL cont.

APPLICATION

This command is used in all position/velocity control tuning applications. For more information on the effectiveness of each gain on system dynamic response, please refer to the *Mx4Pro: Tuning Expert* manual. This manual will help you understand the significance of gains in tuning. Please read this even if you cannot run *Mx4Pro* on your machine because it lacks the DOS operating system.

Command Sequence Example

See `AXMOVE` and `VELMODE`

EXAMPLE

Set the following modified PID gain values for axes 2 and 4:

K_i = 100
K_p = 4000
K_f = 3000
K_d = 2500

K_i = 20
K_p = 8000
K_f = 5500
K_d = 7000

```
CTRL (0xA,100,4000,3000,2500,20,8000,5500,7000)
```

CTRL_KA

FUNCTION Acceleration Feedforward Control Law Parameter

EXECUTION 200 microseconds

SYNTAX `CTRL_KA (n, ka1, ... , ka8)`

USAGE DSPL (Motion), Host (command code: 59h)

ARGUMENTS

n	bit coding of the specified axis(es)
ka_x	unsigned value for Ka gain
	$0 <= ka_x <= 32767$

When used in DSPL, the arguments ka_x may be selected as a variable.

DESCRIPTION

The CTRL_KA command allows the user to program an acceleration feedforward gain for the specified axis(es).

SEE ALSO `CTRL, KILIMIT, OFFSET, OUTGAIN`

EXAMPLE

Program a Ka of 5000 for both axes 1 and 3.

```
CTRL_KA (0x5, 5000, 5000)
```

CUBIC_INT

FUNCTION Start the Internal Cubic Spline Contouring Execution

EXECUTION 100 microseconds

SYNTAX `CUBIC_INT (m, si, n)`

USAGE DSPL (Motion), Host (command code: B1h)

ARGUMENTS

m — specifies the number of points in the cubic spline table to run. Each point is characterized by the position and velocity for only one motor. The maximum number of points is 2,000.

si — specifies the starting index in the table

n — specifies the number of times m points of a spline table will be looped over

$n \leq 32767$

When used in DSPL, arguments m, si, and n may be selected as variables.

Note: n = 0 means run the specified number of points infinite number of times.

CUBIC_INT cont.

DESCRIPTION

This command starts execution of the points stored in the cubic spline table immediately. It takes DSPL (or RTC) approximately 5 ms to interpret this command. After interpretation of this command, DSPL will move on to the next command line. The command sequence for this instruction is as follows:

1) `CUBIC_RATE`
2) `CUBIC_SCALE` ;if necessary
3) `CUBIC_INT`

We assume that user has already downloaded the table points to the cubic spline table location.

Upon execution of a `CUBIC_INT` command, the DSPL program flow will not proceed to a following `CUBIC_INT`, `CUBIC_RATE`, or `CUBIC_SCALE` command until the current `CUBIC_INT` motion is completed. If the command following the `CUBIC_INT` command is not a `CUBIC_INT`, `CUBIC_RATE`, or `CUBIC_SCALE` command, the DSPL program flow will proceed to that command immediately after the `CUBIC_INT` command execution.

SEE ALSO `CLEAR_CUBIC` (*Mx4 User's Guide*), `CUBIC_RATE`, `CUBIC_SCALE`, `CUBIC_TSCALE`

APPLICATION

Refer to Cubic Spline

EXAMPLE

Refer to *Cubic Spline Application Notes.*

CUBIC_RATE

FUNCTION Set Cubic Spline Point Transfer Rate

EXECUTION 50 microseconds

SYNTAX `CUBIC_RATE (m)`

USAGE DSPL (Motion), Host (command code: A1h)

ARGUMENTS

m — parameter coding the value for cubic spline transfer rate. "m" codes the time interval between the adjacent position/velocity points. Its value ranges between 5 and 511 and when divided by 5 it represents the interval in ms. For example, m=5 represents the time interval of 1 ms and m=25 is a 5 ms interval.

When used in DSPL, the argument m may be selected as a variable.

DESCRIPTION

This command sets the point transfer rate for the cubic spline. The "transfer rate" sets the interval between two adjacent points in the ring buffer. The two adjacent points can be spaced anywhere between 1.0 to 102.4 ms. Mx4's cubic spline interpolates between the two adjacent points at 200 us increments. This means for example, Mx4 interpolates 500 points between two adjacent points 100 ms apart. Position and velocity points in the ring buffer are organized similar to the way they are in ordinary contouring. That is, every point is represented by eight bytes **-** four for position and four for velocity.

Since velocity is numerically presented by a 25-bit two's complement number (8 bits (absolute) integer, 16 bits fractional) the upper most significant four bits of 32-bit long velocity are used to code the axes for which the position/velocity points have been specified. For example, the following 32-bit number, **3**0 55 66 77h specifies velocity value 0 55 66 77h in cubic spline interpolation involving axis 1 and axis 2 (i.e., **3** = 0011). Note that the 4-bit axis coding is only used in cubic spline **-** ordinary contouring lacks this feature. Mx4's other contouring feature (i.e., 2nd order) uses the `VECCHG` command to encode the axes involved in a contouring task.

CUBIC_RATE cont.

The contouring strategy can be switched between cubic spline and 2nd order using CUBIC_RATE and BTRATE, respectively. It may take up to 500 ms to execute a CUBIC_RATE. Once a CUBIC_RATE is issued, there is no need to re-issue this command.

The ring buffer breakpoint interrupt cannot detect less than 5 ms worth of points. This imposes a constraint on the minimum number of points for short block transfer rates such as 1 ms. For example, for a 1 ms block transfer rate, a minimum of 5 points in the ring buffer is required.

buffer_break_point(m)	m is number of pos/vel points in ring buffer
for b.t. rate of 1 ms	5 ≤ m ≤ 84 points
for b.t. rate of 5 ms	1 ≤ m ≤ 84 points

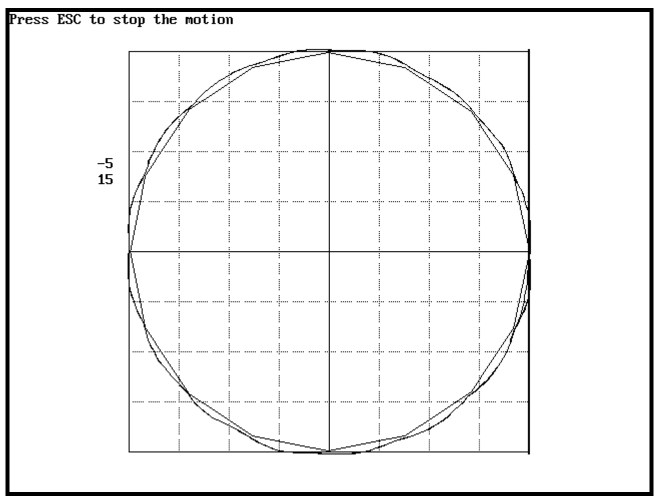
Upon execution of a CUBIC_INT command, the DSPL program flow will not proceed to a following CUBIC_INT, CUBIC_RATE, or CUBIC_SCALE command until the current CUBIC_INT motion is completed. If the command following the CUBIC_INT command is not a CUBIC_INT, CUBIC_RATE, or CUBIC_SCALE command, the DSPL program flow will proceed to that command immediately after the CUBIC_INT command execution.

SEE ALSO EN_BUFBRK, BTRATE, CLEAR_CUBIC (*Mx4 User's Guide*), CUBIC_INT, CUBIC_SCALE

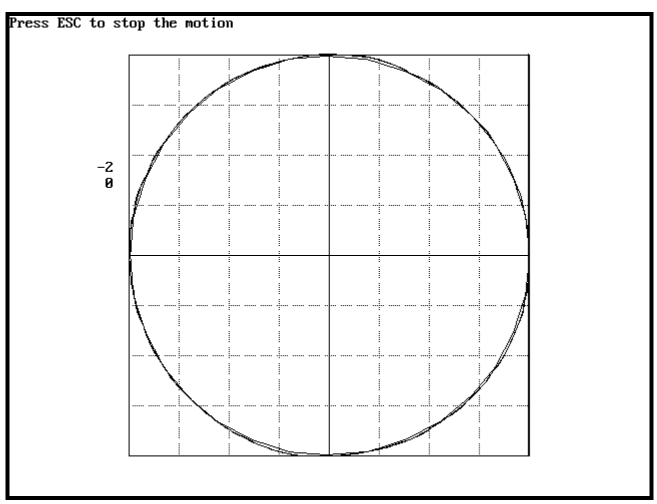
APPLICATION

Refer to *Cubic Spline Application Notes.*

CUBIC_RATE cont.

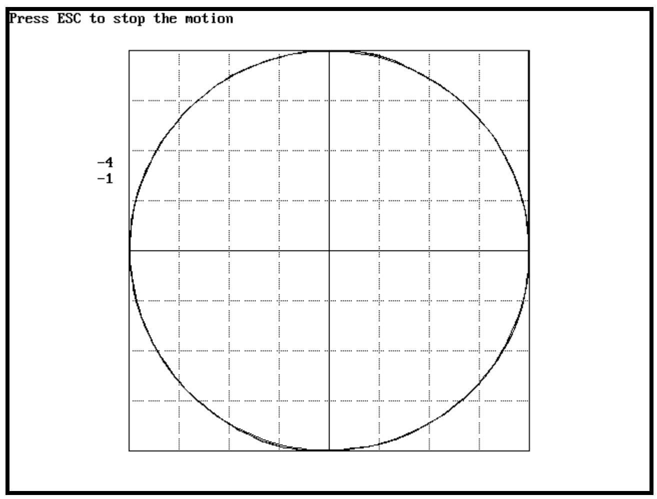


16 points; b.t. rate = 80 ms

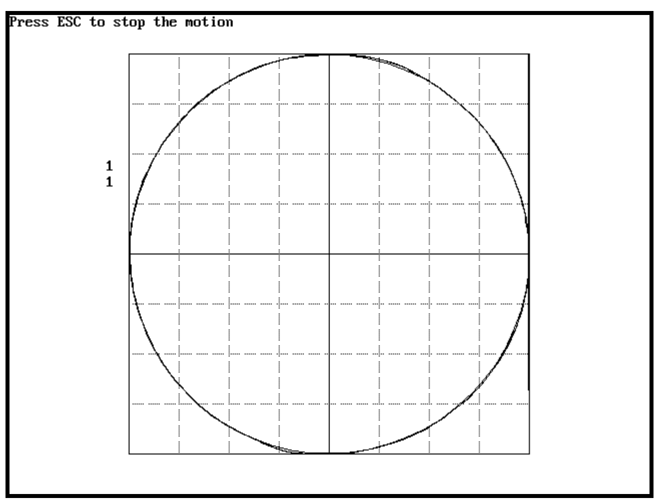


32 points; b.t. rate = 40 ms

CUBIC_RATE cont.



64 points; b.t. rate = 20 ms



128 points; b.t. rate = 10 ms

CUBIC_SCALE

FUNCTION Scales Positions/Velocities, also Shifts Positions

EXECUTION 200 microseconds

SYNTAX `CUBIC_SCALE (n, pv_mult_x, pos_shift_x, ...)`

USAGE DSPL (Motion), Host (command code: B0h)

ARGUMENTS

n — bit coding the axes involved

pv_mult_x — position/velocity scaling multiplier for axis x

$-2 \le pos\_mult_x < 2$

pos_shift_x — position shift for axis x. This is a 32-bit two's complement integer number that transfers the position to a new origin.

When used in DSPL, the arguments pv_mult_x and pos_shift_x may be selected as variables.

DESCRIPTION

This command scales those table points involved in a cubic spline operation. This command also shifts the positions involved by a user defined position shift value.

Upon execution of a `CUBIC_INT` command, the DSPL program flow will not proceed to a following `CUBIC_INT`, `CUBIC_RATE`, or `CUBIC_SCALE` command until the current `CUBIC_INT` motion is completed. If the command following the `CUBIC_INT` command is not a `CUBIC_INT`, `CUBIC_RATE`, or `CUBIC_SCALE` command, the DSPL program flow will proceed to that command immediately after the `CUBIC_INT` command execution.

SEE ALSO `CLEAR_CUBIC` (*Mx4 User's Guide*), `CUBIC_INT`, `CUBIC_RATE`, `CUBIC_TSCALE`

EXAMPLE

Refer to *Cubic Spline Application Notes*

CURR_LIMIT

FUNCTION Set Output Drive Current Limit

EXECUTION 200 microseconds

SYNTAX `CURR_LIMIT (n, clmt1, ... , clmt8)`

USAGE DSPL (Motion), Host (command code: 77h)

ARGUMENTS

n	bit coding of the specified axis(es)
$clmt_x$	unsigned value specifying the current limit percentage $0 \leq clmt_x \leq 100(\%)$

DESCRIPTION

This command sets the current limit for the axes specified. The current limit is defined as a percentage of the maximum desired current (which in turn is defined by the current feedback mechanism). In the case that the current in any phase of a specified axis exceeds the set value, the PWM signals for that axis will turn off for at least one full period and turn on only if the sensed current is reduced below the current limit.

Note: Mx4 with Vx4++ will not execute the `CURR_LIMIT` command if the `VX4_BLOCK` command is active for the axes in question.

SEE ALSO `Vx4_BLOCK`

APPLICATION

See *Vx4++ User's Guide*

EXAMPLE

For current feedback designed for full scale at 10 amps, set current limits of 3 and 4 amps for axes one and two, respectively.

$$(3/10) * 100\% = 30\% \qquad (4/10) * 100\% = 40\%$$

`CURR_LIMIT` (0x3, 30, 40)

CURR_OFFSET

Vx4++ option command

FUNCTION Compensate Current Feedback Offset

EXECUTION 200 microseconds

SYNTAX `CURR_OFFSET (n, val1, ... , val8)`

USAGE DSPL (Motion), Host (command code: 85h)

ARGUMENTS

n	bit coding of the specified axis(es)
val_x	offset value for axis x
	$-32768 \le val_x \le 32767$

When used in DSPL, the argument val_x may be selected as a variable.

DESCRIPTION

The `CURR_OFFSET` command allows the user to compensate for any offset generated by the current feedback path.

Note: Mx4 with Vx4++ will not execute the `CURR_OFFSET` command if the `VX4_BLOCK` command is active for the axes in question.

SEE ALSO `Vx4_BLOCK`

APPLICATION

See *Vx4++ User's Guide*

EXAMPLE

Program an offset compensation value of 2500 for axis one and -1500 for axis four.

```
CURR_OFFSET (0x9, 2500, -1500)
```

CURR_PID

Vx4++ option command

FUNCTION Current Loop Control Law Parameters

EXECUTION 200 microseconds

SYNTAX `CURR_PID (n, par11, ... , par13, ..., par81, ... , par83)`

USAGE DSPL (Motion), Host (command code: 7Bh)

ARGUMENTS

n	bit coding of the specified axis(es)
par_{x1}	unsigned value for K_p gain
par_{x2}	unsigned value for K_i gain
par_{x3}	unsigned value for K_d gain

$0 \le par_{x1,2,3} \le 32767$

DESCRIPTION

This command performs a vector control algorithm combined with a modified PID.

SEE ALSO `CTRL`

APPLICATION

See *Vx4++ User's Guide*

EXAMPLE

Set the following modified current loop PID gain values for axis three.

K_p = 10000
K_i = 20
K_d = 9500

```
CURR_PID (0x4, 10000, 20, 9500)
```

CVEL1, ..., CVEL8 — IDENTIFIER

IDENTIFIER Command Velocity State Variable

USAGE DSPL (PLC, Motion)

DESCRIPTION

A command velocity state variable holds a 25-bit two's complement value (sign extended to 32 bits, the least significant 16 bits represent the fractional portion of the value) that represents the velocity (in encoder edge counts/200µs) that DSPL is commanding the specified axis to reach. For example:

CVEL1 = 000A8000h = 10.5 counts/200µs

Name	Description
`CVEL1`	axis 1 command velocity
`CVEL2`	axis 2 command velocity
`CVELx`	axis x command velocity
.	
.	
`CVEL8`	axis 8 command velocity

SEE ALSO `VEL1`

EXAMPLE

The command velocity state variables can be used as follows:

- as one of the values used in conjunction with a DSPL arithmetic operation:

```
VAR12 = CVEL2 + 0.5
```

- as one of the arguments in a DSPL conditional expression:

```
WAIT_UNTIL(CVEL1 > 1.5)
```

DDAC

FUNCTION Direct DAC Output

EXECUTION 200 microseconds

SYNTAX `DDAC (n, val1, ... , val8)`

USAGE DSPL (Motion), Host (command code: 63h)

ARGUMENTS

n	bit coding for the specified axis(es)
val_x	DAC output voltage for axis x
	$-10.0 \le val_x \le 9.9997$ volts

When used in DSPL, the argument val_x may be selected as a variable.

DESCRIPTION

The `DDAC` command places the axis(es) in open loop, with DAC(x) output voltage determined by the val_x command argument. `DDAC` specifies a bipolar analog signal ranging from -10 to +10 volts with a resolution of approximately 0.3 millivolts.

After execution of a `DDAC` command, in order to return the axis(es) to closed loop operation, a closed-loop command such as `AXMOVE` or `VELMODE` must be executed. The following procedure serves as an example:

1. slow or halt the axis(es) motion:
   -execute `DDAC` with 0v specified

2. minimize built-up following error:
   -execute `POS_PRESET` command

3. return axis(es) to closed loop:
   -execute `AXMOVE` command with target position specified as that used in the preceding `POS_PRESET` command.

SEE ALSO none

DDAC cont.

APPLICATION

This command can be used in applications where the voltage command provides adequate control. Voltage commands can be applied to a torque loop (for torque control applications in robotics) or a velocity loop (to a spindle axis in machine tool applications).

Command Sequence Example

No preparation is required before running this instruction.

EXAMPLE

Output +3.75 volts to the axis 4 DAC.

```
DDAC (0x8,3.75)
```

DELAY

FUNCTION Program Flow Delay

EXECUTION Depends on user arguments

SYNTAX `DELAY (del)`

USAGE DSPL (Motion)

ARGUMENTS

del — value specifying the number of 200µs intervals to delay

$0 \le del \le 65535$ (200µs intervals)

When used in DSPL, the argument del may be selected as a variable.

DESCRIPTION

DSPL Motion program flow stops at the `DELAY` command for the specified amount of time.

SEE ALSO `WAIT_UNTIL`

APPLICATION

See *Application Notes*

EXAMPLE

Set a delay of 0.400 seconds.

0.400 / (200 e-006) = 2000

```
DELAY (2000)
```

DISABL_INT

FUNCTION Disable Interrupts

EXECUTION 10 microseconds

SYNTAX `DISABL_INT (n, m1, ... , m8)`

USAGE DSPL (Motion), Host (command code: 64h)

ARGUMENTS

n	bit coding of the specified axis(es)
m_x	bit coding of the interrupts to disable for axis x (setting a bit to 1 indicates disabling an interrupt)

bit 7	:	not used
bit 6	:	motion complete
bit 5	:	index
bit 4	:	probe
bit 3	:	position breakpoint
bit 2	:	following error
bit 1	:	following error / halt
bit 0	:	buffer breakpoint

DESCRIPTION

This command disables some or all of the servo control card interrupts.

SEE ALSO `DISABLE2_INT, EN_BUFBRK, EN_PROBE, EN_ERR, EN_ERRHLT, EN_INDEX, EN_MOTCP, EN_POSBRK`

APPLICATION

This command may be used in conjunction with all applications in which only a few interrupts are needed to be enabled. Also, a few enabled interrupts may have to be disabled based on external events.

Command Sequence Example

No preparation is required before running this instruction.

DISABL_INT cont.

EXAMPLE

Disable the previously enabled axis 1 following error and axis 3 index pulse interrupts.

```
DISABL_INT (0x5,0x04,0x20)
```

DISABL2_INT

FUNCTION Disable Interrupts

EXECUTION 10 microseconds

SYNTAX `DISABL2_INT (n, m1, ... , m8)`

USAGE DSPL (Motion), Host (command code: 5Ah)

ARGUMENTS

n	bit coding of the specified axis(es)
m_x	bit coding of the interrupts to disable for axis x (setting a bit to 1 indicates disabling an interrupt)

bit 7	:	not used
bit 6	:	not used
bit 5	:	not used
bit 4	:	not used
bit 3	:	not used
bit 2	:	not used
bit 1	:	not used
bit 0	:	encoder fault [EN_ENCFLT]

DESCRIPTION

This command disables the selected enabled interrupts.

SEE ALSO `DISABL_INT, EN_ENCFLT`

APPLICATION

This command may be used in conjunction with all applications in which only a few interrupts are needed to be enabled. Also, a few enabled interrupts may have to be disabled based on external events.

Command Sequence Example

No preparation is required before running this instruction.

DISABL2_INT cont.

EXAMPLE

Disable the previously enabled axis 1, axis 3, and axis 4 encoder fault [EN_ENCFLT] interrupts.

```
DISABL2_INT (0xd, 0x01, 0x01, 0x01)
```

ELSE

FUNCTION Else Condition in IF-(then)-(ELSE)-ENDIF Structure

EXECUTION 10 microseconds

SYNTAX

```
IF (conditional expression)
   program code to execute if the IF condition is True
ELSE
   program code to execute if the IF condition is False
ENDIF
```

USAGE DSPL (PLC, Motion)

ARGUMENTS

none

DESCRIPTION

The IF-(then)-ELSE structure is used for conditional program execution. The ELSE operand allows selective program execution as a result of a False IF conditional expression.

SEE ALSO `IF, ENDIF`

APPLICATION

See *Application Notes*

EXAMPLE

Preset the position of axis one to 100 counts if the command position of axis two is > 1000 counts; otherwise preset the position of axis one to 200 counts.

```
IF (CPOS2 > 1000)
        POS_PRESET (0x1,100)
ELSE
        POS_PRESET (0x1,200)
ENDIF
```

EN_BUFBRK

FUNCTION Enable Buffer Breakpoint Interrupt

EXECUTION 10 microseconds

SYNTAX `EN_BUFBRK (buffbrk)`

USAGE DSPL (Motion), Host (command code: 61h)

ARGUMENTS

buffbrk — a positive value which represents the delta position for the remaining number of bytes in the ring buffer. Since each contouring point requires 8 bytes, this number must be multiplied by 8 to indicate the real number of bytes left in the ring buffer.

$1 \le$ buffbrk ≤ 84 contouring data points

DESCRIPTION

This command will cause an interrupt when the number of contouring data points in the contouring ring buffer falls below a preset breakpoint. The buffer breakpoint interrupt status will appear in bit 0 of the DPR interrupt flag location [Mx4:7FEh] [Mx4 Octavia:1FFEh]. This bit gets set if a buffer breakpoint interrupt occurs.

SEE ALSO `DISABL_INT`

APPLICATION

This command must be used in both 2nd order and cubic spline contouring applications. To maintain continuity in a contouring application, Mx4 must be constantly updated by the host processor with a set of new (position/velocity) points on the contour. Since no application can afford to run out of points, the host must set the buffer breakpoint interrupt to a value such that running the remaining points (what is left in the ring buffer) will give the host enough time to update the buffer. For slower hosts, the argument for this command must be relatively larger.

EN_BUFBRK cont.

Command Sequence Example

`MAXACC ( )`	;set the maximum accel. so system can be stopped
`CTRL ( )`	;set the gains
`KILIMIT ( )`	
.	;load the ring buffer with contouring points,
.	;(position and speed)
`BTRATE ( )`	;set the 2nd order contouring block transfer rate to 5, 10, 15 or 20 ms
`EN_BUFBRK ( )`	;set the breakpoint in buffer
.	
.	
`START (n)`	;start contouring

EXAMPLE

Enable a contouring ring buffers breakpoint interrupts for the case that the number of segment move commands in the ring buffer falls below 30.

```
EN_BUFBRK (30)
```

ENCOD_MAG

Vx4++ option command

FUNCTION Define Encoder Line Count, Motor Poles, Commut. Option

EXECUTION 200 microseconds

SYNTAX `ENCOD_MAG (n, p1`$_1$`, p2`$_1$`, p3`$_1$`, ... , p1`$_4$`, p2`$_4$`, p3`$_4$`)`

USAGE DSPL (Motion), Host (command code: 80h)

ARGUMENTS

n — bit coding of the specified axis(es)

$p1_x$ — number of encoder lines/rev on axis x

$0 \le p1_x \le 65535$

$p2_x$ — number of motor poles on axis x

$0 \le p2_x \le 256$

$p3_x$ — brushless DC commutation option

$p3_x = 0$: brushtype DC or AC induction motor tech
$p3_x = 0$: comm option 0
$p3_x = 1$: comm option 1

DESCRIPTION

The Vx4++ option card interfaces to the motors with any number of magnetic poles and encoders with any number of encoder pulse numbers. An example of this is a brushless DC machine with eight poles, a 1,000 line encoder, and hall sensors mounted in a special configuration. This command allows the user to define the encoder, commutation, and motor pole parameters for the specified axis(es).

Note: Mx4 with Vx4++ will not execute the `ENCOD_MAG` command if the `VX4_BLOCK` command is active for the axes in question.

SEE ALSO `VX4_BLOCK`

ENCOD_MAG cont.

APPLICATION

See *Vx4++ User's Guide*

EXAMPLE

Axis four is an AC induction motor with a 1024 line encoder and 4 motor poles.

```
ENCOD_MAG (0x8, 1024, 4, 0)
```

ENDIF

FUNCTION Designates End Of IF-(then)-(ELSE)-ENDIF Structure

EXECUTION 10 microseconds

SYNTAX

```
IF (conditional expression)
    program code to execute if the IF condition is True
ELSE
    program code to execute if the IF condition is False
ENDIF
```

USAGE DSPL (PLC, Motion)

ARGUMENTS

none

DESCRIPTION

The IF-(then)-ELSE structure is used for conditional program execution. ENDIF designates the last line of the IF-(then)-ELSE structure. An ENDIF statement must be included with every IF statement.

SEE ALSO `IF, ELSE`

APPLICATION

See *Application Notes*

EXAMPLE

Preset the position of axis one to 100 if `VAR1` is equal to 0. If `VAR1` is not equal to 0 and `VAR2` is equal to 1, preset the axis one position to 200.

```
IF (VAR1 == 0)
        POS_PRESET (0x1,100)
ELSE
        IF (VAR2 == 1)
                POS_PRESET (0x1,200)
        ENDIF
ENDIF
```

EN_ENCFLT

FUNCTION Encoder Fault Interrupt

EXECUTION 50 microseconds

SYNTAX `EN_ENCFLT (m, n, fer1, ... , fer8)`

USAGE DSPL (Motion), Host (command code: 58h)

ARGUMENTS

m	bit coding of the axes interrupt condition (see Description)
n	bit coding of the specified axis(es)
fer_x	unsigned following error value for axis x 0 <= fer_x <= 65535 counts

DESCRIPTION

This command enables the encoder fault interrupt for the specified axes.

With the respective axis bit of argument m equal to 0, the encoder fault interrupt is triggered for the axis in question if,

1. abs[following error] > $ferr_x$ threshold, and
2. hardware encoder status bit is set

With the respective axis bit of argument m equal to 1, the encoder fault interrupt is triggered for the axis in question if,

1. abs[following error] > $ferr_x$ threshold

If an encoder fault interrupt condition is present for an axis, the axis will be put into open loop with DAC output of 0 volts, and an interrupt will be generated. If, however, the axis in question is already in open

EN_ENCFLT cont.

loop prior to the interrupt condition, an interrupt will be generated but no action will be taken (ie: DAC voltage is unaffected).

The encoder fault interrupt is sustained until the EN_ENCFLT command is reissued to the Mx4. Reissuing the EN_ENCFLT command also allows the affected axis(es) to be put back into closed loop following the execution of the command.

The hardware encoder status bits are reported to the lower nibble of DPR location 113h (see Mx4 DPR Organization). A set bit indicates that Mx4 has detected an encoder hardware failure. Mx4 reports an “encoder status” error if for the axis in question,

1. the encoder feedback to Mx4 is losing encoder pulses or one of the encoder signals (A or B) actively toggles while the other one is inactive.

The DPR interrupt status locations 009h (bit 4) and 00Eh record the occurrence and source of this interrupt, respectively. Bit 6 of DPR location [Mx4:7FEh] [Mx4 Octavia:1FFEh] is also set.

SEE ALSO `DISABL2_INT`

APPLICATION

A necessary diagnostic feature for all servo control applications.

Command Sequence Example

No preparation is required before running this instruction.

EXAMPLE

Enable the encoder fault interrupt for both axis 3 and axis 4. Set the following error threshold at 500 counts, using the encoder hardware status bits in the interrupt conditions.

```
EN_ENCFLT (0xc, 0xc, 500, 500)
```

EN_ERR

FUNCTION Enable Following Error Interrupt

EXECUTION 50 microseconds

SYNTAX `EN_ERR (n, fer1, ... , fer8)`

USAGE DSPL (Motion), Host (command code 67h)

ARGUMENTS

n — bit coding of the specified axis(es) for which the interrupt is enabled

fer_x — unsigned following error value for axis x

$0 \le fer_x \le 65535$ counts

When used in DSPL, the argument fer_x may be selected as a variable.

DESCRIPTION

Upon the execution of this command, if at any time the following error for a specified axis exceeds its programmed value, the servo control card will generate an interrupt. This condition is recorded in DPR interrupt status register location 000h. The DPR status register location 02h will identify the axis(es) responsible. Bit 1 of DPR location [Mx4:7FEh] [Mx4 Octavia:1FFEh] is also set.

The interrupt condition is also axis bit-coded in the DSPL `FERR_REG` bit register.

Note: `EN_ERR` is not disabled after it occurs. The host is responsible for disabling the interrupt.

SEE ALSO `DISABL_INT, EN_ERRHLT`

APPLICATION

This command may be used in all applications for two reasons. First, `EN_ERR` reports a run-away or any other out-of-control condition.

EN_ERR cont.

Second, it makes sure that position error is within a specified tolerance (i.e. the value in argument fer_x.)

Command Sequence Example

No preparation is required before running this instruction.

EXAMPLE

Set a `EN_ERR` interrupt value of 200 encoder counts for axis 1.

```
EN_ERR (0x1, 200)
```

EN_ERRHLT

FUNCTION Enable Following Error Interrupt and Halt

EXECUTION 50 microseconds

SYNTAX EN_ERRHLT (n, fer_1, ... , fer_8)

USAGE DSPL (Motion), Host (command code: 66h)

ARGUMENTS

n	bit coding of the specified axis(es) for which the interrupt is enabled
fer_x	unsigned following error value for axis x
	$0 \le fer_x \le 65535$ counts

When used in DSPL, the argument fer_x may be selected as a variables.

DESCRIPTION

Upon execution of this command, if at any time the following error for a specified axis exceeds it's programmed value, the system will halt and generate an interrupt. The halt brings the motion of the axis in question to a stop using the programmed maximum acceleration rate. This interrupt condition is recorded in DPR interrupt status register location 000h. The DPR status register location 001h reveals the axis(es) responsible. Bit 1 of DPR location [Mx4:7FEh] [Mx4 Octavia:1FFEh] is also set.

The interrupt condition is also axis bit-coded in the DSPL `FERRH_REG` bit register.

Note 1: `EN_ERRHLT` will be ignored if the respective axis abort maximum acceleration is zero.

Note 2: `EN_ERRHLT` is not disabled after it occurs. The host is responsible for disabling the interrupt.

EN_ERRHLT cont.

SEE ALSO `DISABL_INT, EN_ERR, ESTOP_ACC`

APPLICATION

Applications of this command are similar to `EN_ERR`. However, as a result of this command's interrupt, the system will come to a stop. Stop trajectory uses the programmed abort maximum acceleration. Please see `ESTOP_ACC`. Please note that this command is not appropriate to prevent system run-away in case of encoder loss, since in the absence of the encoder, the system cannot be stopped reliably.

Command Sequence Example

```
ESTOP_ACC ( )   ;set the maximum accel. so system can be stopped
CTRL ( )        ;these instructions enable system to stop motion
KILIMIT ( )     ;set gains
.
.
EN_ERRHLT ( )
```

EXAMPLE

Enable a following error/halt interrupt for axis 1, 2 and 3 with a threshold of 100, 120 and 200 counts, respectively.

```
EN_ERRHLT (0x7,100,120,200)
```

EN_INDEX

FUNCTION Enable Index Pulse Interrupt

EXECUTION 200 microseconds

SYNTAX `EN_INDEX (n)`

USAGE DSPL (Motion), Host (command code: 69h)

ARGUMENTS

n bit coding the ***only*** axis for which the interrupt is enabled

DESCRIPTION

Upon the execution of this command, the servo control card will search for the first index pulse edge from the specified axis. The pulse edge generates an interrupt and registers the actual position for all axes in DPR locations 103h - 112h. The DPR interrupt status register locations 000h and 003h record the occurrence and source of this interrupt. Bit 1 of DPR location [Mx4:7FEh] [Mx4 Octavia:1FFEh] is also set.

The interrupt condition is also axis bit-coded in the DSPL `INDEX_REG` bit register.

Note 1: Only one index pulse can generate an interrupt at any given time. The `EN_INDEX` command enables the index pulse interrupt for the axis specified and automatically disables the previous one (if any).

Note 2: The `EN_INDEX` and `EN_PROBE` commands CAN BE ENABLED simultaneously.

SEE ALSO `DISABL_INT`, `POS_PRESET`, `POS_SHIFT`

EN_INDEX cont.

APPLICATION

This command is used in homing applications. As a result of this instruction, Mx4 will start searching for the first index pulse edge. Upon the detection of an index pulse edge, position of the axis is immediately recorded. This instruction must be used in conjunction with `POS_PRESET` to perform homing for linear table (or other index-based) position calibration.

Command Sequence Example

No preparation is required before running this instruction.

EXAMPLE

Enable the index pulse interrupt for axis 4.

```
EN_INDEX (0x8)
```

EN_MOTCP

FUNCTION Enable Motion Complete Interrupt

EXECUTION 10 microseconds

SYNTAX `EN_MOTCP (n)`

USAGE DSPL (Motion), Host (command code: 65h)

ARGUMENTS

n bit coding of the specified axis(es) for which the interrupt is enabled

DESCRIPTION

This command enables the motion complete interrupt for the axes specified. The motion complete interrupt is generated when any closed loop motion other than ring buffer 2nd order or ring buffer cubic spline contouring comes to a stop. The DPR interrupt status register locations 000h and 005h record the occurrence and source of this interrupt. Bit 1 of DPR location [Mx4:7FEh] [Mx4 Octavia:1FFEh] is also set.

The interrupt condition is also bit-coded in the DSPL `MOTCP_REG` bit register.

Note: `EN_MOTCP` is not disabled after it occurs. The host is responsible for disabling the interrupt.

SEE ALSO `DISABL_INT`

APPLICATION

In any application that a new routine must run based on the end of a motion, this command informs the host of motion completion. An example of such an application is milling in which the spindle and z-axes will start moving only when the x-y table has moved to a target position.

Command Sequence Example

See `AXMOVE` and `STOP`

EN_MOTCP cont.

EXAMPLE

Enable the motion complete interrupt for all four axes.

```
EN_MOTCP (0xF)
```

EN_POSBRK

FUNCTION Enable Position Breakpoint Interrupt

EXECUTION 100 microseconds

SYNTAX `EN_POSBRK (n, pos`$_1$`, ... , pos`$_8$`)`

USAGE DSPL (Motion), Host (command code: 6Bh)

ARGUMENTS

n	bit coding of the specified axis(es) for which the interrupt is enabled
pos_x	position breakpoint position value for axis x $-2147483648 \le pos_x \le 2147483647$ counts

When used in DSPL, arguments `pos`$_x$ may be selected as a variable.

DESCRIPTION

This command enables the position breakpoint interrupt for the axes specified. The position breakpoint interrupt is generated when the actual position, for a specified axis, passes the programmed breakpoint. The DPR interrupt status register locations 000h and 004h record the occurrence and source of this interrupt. Bit 1 of DPR location [Mx4:7FEh] [Mx4 Octavia:1FFEh] is also set.

The interrupt condition is also axis bit-coded in the DSPL `POSBRK_REG` bit register.

Note 1: The position breakpoint is calculated as the absolute distance from the present position (position at the moment at which the `EN_POSBRK RTC` is interpreted) to the position breakpoint value entered. The breakpoint interrupt is set when the axis in question travels (in either direction) a distance equal to the calculated absolute distance.

EN_POSBRK cont.

Note 2: `EN_POSBRK` is automatically disabled after the breakpoint interrupt is generated. To activate this interrupt again, the host must issue a new `EN_POSBRK` command.

Note 3: `POS_PRESET` and `POS_SHIFT` will automatically disable the position breakpoint interrupt. The user is responsible for re-enabling the intterupt.

SEE ALSO `DISABL_INT, POS_PRESET, POS_SHIFT`

APPLICATION

This instruction may be used in applications such as robotics, indexing machine tools, etc. The CPU must be notified that the system has passed an intermediate position. Based on this interrupt, the CPU will execute a task. For example, in a robotics painting application, the paint mixture may have to change based on the robot's arm location.

Command Sequence Example

```
MAXACC ( )          ;set the maximum accel. so system can be stopped
CTRL ( )            ;set the gains
KILIMIT ( )
OUTGAIN ( )
```

EXAMPLE

Enable a breakpoint interrupt with a value of 60,000 counts for axis 1 and 500,000 for axis 2.

```
EN_POSBRK (0x3,60000,500000)
```

EN_PROBE

FUNCTION Enable General Purpose External Interrupt

EXECUTION 200 microseconds

SYNTAX `EN_PROBE (m)`

USAGE DSPL (Motion), Host (command code: 6Ch)

ARGUMENTS

m bit coding of the only *EXTx input signal enabled

[Mx4]

m=1h	:	from *EXT1
m=2h	:	from *EXT2

[Mx4 Octavia]

m=1h	:	from *EXT1
m=2h	:	from *EXT2
m=4h	:	from *EXT3
m=8h	:	from *EXT4

DESCRIPTION

Upon the execution of this command, the servo control card will search for the first *EXTx pulse edge. The pulse edge generates an interrupt, and registers the actual position for all axes in DPR locations 0A7h-0B6h. (The hand shaking bytes are 0C8h and 0D0h for Mx4 and host, respectively.) DPR interrupt status register locations 000h and 006h record the occurrence and source of this interrupt. Bit 1 of DPR location [Mx4:7FEh] [Mx4 Octavia:1FFEh] is also set.

The interrupt condition is also axis bit-coded in the DSPL `PROBE_REG` bit register.

EN_PROBE cont.

Note 1: Only one general purpose external interrupt can generate an interrupt at any given time. The `EN_PROBE` command enables the external interrupt specified and automatically disables the previous one (if any).

Note 2: The `EN_PROBE` and `EN_INDEX` can be enabled simultaneously.

SEE ALSO `DISABL_INT, ESTOP_ACC`

APPLICATION

This instruction is useful in probing applications. Since `EN_PROBE` registers all positions when an interrupt occurs (falling pulse edge is detected) it can be used in accurate recording of surface dimensions by a probe.

Command Sequence Example

```
CTRL ( )          ;these instructions enable system to stop motion
KILIMIT ( )
.
.
EN_PROBE ( )
END
```

EXAMPLE

Enable the *EXT2 external interrupt.

```
EN_PROBE (0x2)
```

ERR1, …, ERR8 — IDENTIFIER

IDENTIFIER Following Error State Variable

USAGE DSPL (PLC, Motion)

DESCRIPTION

A following error state variable holds a 32-bit two's complement integer value that represents the difference between the current position and the actual position (in encoder edge counts) of the specified axis.

Name	Description
`ERR1`	axis 1 following error
`ERR2`	axis 2 following error
`ERRx`	axis x following error
.	
.	
`ERR8`	axis 8 following error

SEE ALSO `CPOS1, INDEX_POS1, POS1, PROBE_POS1`

EXAMPLE

The following error state variables can be used as follows:

- as one of the values used in conjunction with a DSPL arithmetic operation:

```
VAR1 = ERR4 / VAR3
```

- as one of the arguments in a DSPL conditional expression:

```
IF(ERR3 <= VAR2)
```

ESTOP_ACC

FUNCTION Abort Motion Maximum Acceleration

EXECUTION 100 microseconds

SYNTAX `ESTOP_ACC (n, acc`$_1$`, ... , acc`$_8$`)`

USAGE DSPL (Motion), Host (command code: 86h)

ARGUMENTS

n	bit coding of the specified axis(es) for which the interrupt is enabled
acc_x	unsigned value specifying the maximum halting acceleration (deceleration) for axis x $0 \le acc_x \le 1.999969$ counts/(200µs)^2

Note: Acceleration is partitioned into 1 bit integer, 15 bits fraction.

When used in DSPL, argument acc_x may be selected as a variable.

DESCRIPTION

This command specifies the maximum halting acceleration (deceleration) for the axes specified. The maximum acceleration values are used in the following cases: `EN_ERRHLT`, and `ESTOP_ACC`.

Note: `ESTOP_ACC` will be ignored if the specified argument is zero.

SEE ALSO `EN_ERRHLT`, `MAXACC`, `STOP`, `VELMODE`

ESTOP_ACC cont.

APPLICATION

This command sets the maximum possible deceleration for a mechanical actuator. This RTC is used to set the deceleration rate for an emergency case. In contrast to `MAXACC`, `ESTOP_ACC` provides a sharper deceleration such that the entire system comes to a stop as rapidly as possible. Please remember that the `STOP` and `VELMODE RTCs` use `MAXACC` for their acceleration/deceleration.

Command Sequence Example

```
ESTOP_ACC ( )   ;set the abort maximum acceleration
CTRL ( )        ;make sure the system is in closed loop
EN_ERRHLT ( )   ;set the maximum tolerance for the following error
                ;if the following error exceeds the ABORTACC
                ;parameter, the system will stop immediately
```

EXAMPLE

Set an abort motion maximum acceleration for axes 2 and 3 of 0.5 encoder counts/(200 µsec)^2.

```
ESTOP_ACC (0x6,0.5,0.5)
```

ESTOP_REG, FERR_REG, FERRH_REG, INDEX_REG, MOTCP_REG, OFFSET_REG, POSBRK_REG, PROBE_REG IDENTIFIER

IDENTIFIER DSPL interrupt registers

USAGE DSPL (PLC, Motion)

DESCRIPTION

The status of a variety of Mx4 interrupt conditions is available to the DSPL programmer. All of the DSPL interrupt bit registers, with the exception of `ESTOP_REG`, are 16-bit registers (bit 0-15) that specify the axis(es) responsible for the interrupt. The least significant four bits of each of these registers follow an LSB (axis 1), MSB (axis 8) format (the most significant 8 bits are unused). For example:

bit 0:	Axis 1 interrupt
bit 1:	Axis 2 interrupt
bit 2:	Axis 3 interrupt
bit 3:	Axis 4 interrupt
bit 4:	Axis 5 interrupt
bit 5:	Axis 6 interrupt
bit 6:	Axis 7 interrupt
bit 7:	Axis 8 interrupt

Since there is only one `ESTOP` signal for all four (8) axes, `ESTOP_REG` is a single-bit (bit 0) register (the most significant 15 bits are unused). In all of the interrupt registers, a set bit (bit = 1) indicates an interrupt.

The bit register may be used with the bitwise operators in conditional expressions within the `DSPL IF`, `WHILE` and `WAIT_UNTIL` conditional structures. The user defined bit mask used in conjunction with the bitwise operator & must follow the format 0x????, where ???? is a 16-bit hexadecimal value. For example, a mask value of 0x0006 will mask out all bits except bits 1 and 2.

ESTOP_REG, FERR_REG, FERRH_REG, INDEX_REG, MOTCP_REG, OFFSET_REG, POSBRK_REG, PROBE_REG cont.

IDENTIFIER

Name	**Bit Values**	**Description**

The `ESTOP_REG` interrupt bit is set if an emergency stop is being signaled.

`ESTOP_REG`	bits 0	Emergency stop interrupt
	bits 1 - 15	unused

An `FERR_REG` interrupt bit is set if the following error for a specified axis exceeds a programmed value.

`FERR_REG`	bits 0 - 7	Following error interrupt
	bits 8 - 15	unused

An `FERRH_REG` interrupt bit is set if the following error for a specified axis exceeds a programmed value. The system is halted.

`FERRH_REG`	bits 0 - 7	Following error & halt interrupt
	bits 8 - 15	unused

An `INDEX_REG` interrupt bit is set when an index pulse edge is reached.

`INDEX_REG`	bits 0 - 7	Index pulse interrupt
	bits 8 - 15	unused

A `MOTCP_REG` interrupt bit is set when any closed loop motion comes to a stop.

`MOTCP_REG`	bits 0 - 7	Motion complete interrupt
	bits 8 - 15	unused

ESTOP_REG, FERR_REG, FERRH_REG, INDEX_REG, MOTCP_REG, OFFSET_REG, POSBRK_REG, PROBE_REG cont. IDENTIFIER

An OFFSET_REG interrupt bit is set when offset tuning has completed.

OFFSET_REG	bits 0 - 7	Offset finished interrupt
	bits 8 - 15	unused

A POSBRK_REG interrupt bit is set when the actual position for a specified axis has passed a certain point.

POSBRK_REG	bits 0 - 7	Position breakpoint interrupt
	bits 8 - 15	unused

A PROBE_REG interrupt bit is set when the first *EXT pulse edge is found.

[Mx4]

PROBE_REG	bits 0 - 1	External probe interrupt
	bits 8 - 15	unused

[Mx4 Octavia]

PROBE_REG	bits 0 - 3	External probe interrupt
	bits 8 - 15	unused

SEE ALSO ~, &, AND, OR, IF, WHILE, WAIT_UNTIL

EXAMPLE

The conditional expression in the DSPL IF statement below will evaluate to TRUE if bit 0 or 2 is set (bit = 1) in the motion complete interrupt register:

```
IF (MOTCP_REG & 0x0005)
```

FLUX_CURRENT

Vx4++ option command

FUNCTION Set Field Compensation Or Flux Value

EXECUTION 200 microseconds

SYNTAX `FLUX_CURRENT (n, fval`$_1$`, ... , fval`$_8$`)`

USAGE DSPL (Motion), Host (command code: 79h)

ARGUMENTS

n — bit coding of the specified axis(es)

$fval_x$ — for AC induction motor, defines a bipolar flux value for the field producing component of the current

$-32768 \leq fval_x \leq 32767$

for brushless DC motor, defines a unipolar field compensation parameter

$0 \leq fval_x \leq 65535$

When used in DSPL, the argument $fval_x$ may be selected as a variable.

DESCRIPTION

The `FLUX_CURRENT` command defines motor technology-dependent parameters. If the axis in question is an AC induction motor, the command defines a bipolar flux value for the field producing component of the current. If the axis is a brushless DC motor, the command sets a unipolar field compensation parameter.

Note: The `FLUX_CURRENT` command does not need to be programmed for brushtype DC motors.

SEE ALSO none

APPLICATION

See *Vx4++ User's Guide*

FLUX_CURRENT cont.

EXAMPLE

Set a flux value or -5000 for axis one (AC induction motor) and a field compensation value of 1300 for axis two (brushless DC motor).

```
FLUX_CURRENT (0x3, -5000, 1300)
```

FRAC

FUNCTION Extract the Fractional Portion of a Constant or a Variable Value.

EXECUTION 10 microseconds

SYNTAX `FRAC(valu)` *or* `-FRAC(valu)`

USAGE DSPL (PLC, Motion)

ARGUMENTS

valu — A constant real number
or a
variable (`VAR1` through `VAR128`)

DESCRIPTION

This function extracts the fractional portion of a constant or a variable value. The fractional portion of a number consists of all of the digits to the right of the decimal point. The returned value will therefore always have an absolute value that is less than 1. If a minus sign appears to the left of the `FRAC` function, the fractional portion of *valu* is multiplied by -1.

Note: This function can only be used with a variable assignment statement. For example:

```
VAR55 = FRAC(17.283)
```

SEE ALSO `ABS, INT, SIGN, SQRT`

EXAMPLE

The first example extracts the fractional portion of the value stored in `VAR27`, and stores the result in `VAR18`:

```
VAR18 = FRAC(VAR27)
```

The second example finds the fractional portion of -882.619 and stores the negated result (0.619) in `VAR38`:

```
VAR38 = -FRAC(-482.619)
```

GEAR

FUNCTION Electronics Gear On

EXECUTION 200 microseconds

SYNTAX `GEAR (n, m, r1, ... , r8)`

USAGE DSPL (Motion), Host (command code: 09Ch)

ARGUMENTS

n	bit coding the ONLY axis as LEADER gear
m	bit coding the axis(es) as FOLLOWER gears
r_x	gear ratio between master and slave $-256 \leq ratio_x < 255.999$ minimum gear ratio is +/- 1/128

When used in DSPL, argument r_x may be selected as variable.

DESCRIPTION

This command emulates the mechanical gear function. The follower follows the leader with the gear ratio specified by r_x. Upon receiving this command, the electronic gearing is engaged at once.

SEE ALSO `GEAR_OFF, GEAR_POS, GEAR_PROBE`

APPLICATION

See *Application Notes*

EXAMPLE

Axis 2 is a slave axis to axis 1 with a gear ratio of 2.5.

```
GEAR (0x1,0x2,2.5)
```

GEAR_OFF

FUNCTION Electronics Gear Off

EXECUTION 10 microseconds

SYNTAX `GEAR_OFF (n)`

USAGE DSPL (Motion), Host (command code: 09Fh)

ARGUMENTS

n bit coding of the FOLLOWER axis(es) to be disengaged

DESCRIPTION

This command disengages the specified follower axes at once.

SEE ALSO `GEAR, GEAR_POS, GEAR_PROBE`

APPLICATION

See *DSPL Application Notes*

EXAMPLE

Axis 1 is the leader, axis 3 and 4 are the followers (slaves). Disengage only axis 4.

```
GEAR_OFF (0x8)
```

GEAR_OFF_ACC

FUNCTION Turns Electronic Gearing Off and Halt Slave(s)

EXECUTION 50 microseconds

SYNTAX `GEAR_OFF_ACC (n)`

USAGE DSPL (Motion), Host (command code: A0h)

ARGUMENTS

n bit coding of the axis to be disengaged

DESCRIPTION

This command disengages the system that was under master slave control. The slave axes will come to a complete stop at the maximum acceleration rate specified by `MAXACC` command.

SEE ALSO `GEAR, GEAR_OFF, GEAR_POS, GEAR_PROBE, SYNC`

APPLICATION

Axis 1 is the leader, axis 3 and 4 are the followers (slaves). Disengage only axis 4.

```
GEAR_OFF_ACC (0x8)
```

GEAR_POS

FUNCTION Electronics Gear On at a Specified Leader Position

EXECUTION 200 microseconds

SYNTAX `GEAR_POS (n, m, r1, tp1, ... , r8, tp8)`

USAGE DSPL (Motion), Host (command code: 09Dh)

ARGUMENTS

n — bit coding of the ONLY axis as LEADER gear

m — bit coding of the FOLLOWER axis(es)

r_x — gear ratio between leader and follower(s) ($ratio_x$: 1)

$-256 \le ratio_x < 255.999$

minimum gear ratio is +/- 1/128

tp_x — leader axis position value at which the electronic gearing engages for the specified axis(es)

$-2147483648 \le tp_x \le 2147483647$

When used in DSPL, arguments r_x and tp_x may be selected as variables.

DESCRIPTION

This command emulates a mechanical gear function. The follower follows the leader with the gear ratio specified by r_x. Upon receiving this command, the electronic gearing starts engaging at the specified master position (tp_x).

SEE ALSO `GEAR`, `GEAR_OFF`, `GEAR_PROBE`

APPLICATION

See *DSPL Application Notes*

GEAR_POS cont.

EXAMPLE

Axes 3 and 4 should follow axis 2 with gear ratios 2.0 and 4.0, respectively. Both axes three and four should “engage” when axis 2 position is equal to 10,500 counts.

```
GEAR_POS (0x2,0xC,2.0,10500,4.0,10500)
```

GEAR_PROBE

FUNCTION Electronics Gear On After Probe Input

EXECUTION 200 microseconds

SYNTAX `GEAR_PROBE (n, m, q, r1, ... , r8)`

USAGE DSPL (Motion), Host (command code: 09Eh)

ARGUMENTS

n	bit coding the ONLY axis as LEADER gear
m	bit coding the FOLLOWER axis(es)
q	the *EXTx probe input to be used

[Mx4]
q = 01h : *EXT1
q = 02h : *EXT2

[Mx4 Octavia]
q = 01h : *EXT1
q = 02h : *EXT2
q = 03h : *EXT3
q = 04h : *EXT4

r_x gear ratio between master and slave(s)

$-256 \leq ratio_x < 255.999$
minimum gear ratio is +/- 1/128

When used in DSPL, argument r_x may be selected as variable.

DESCRIPTION

This command emulates the mechanical gear function. The follower follows the leader with the gear ratio specified by r_x. The `GEAR_PROBE` command engages the mechanical gear function for selected master and slave axes after the specified external signal (*EXTx) is activated.

GEAR_PROBE cont.

Note 1: Execution of the `GEAR_PROBE` command will disable any previously enabled `EN_PROBE` interrupt. Probe (*EXT1,2,3,4) activation does ***not*** generate an interrupt with the `GEAR_PROBE` command.

Note 2: Activation of *ESTOP during a GEAR operation will halt the master axis, and subsequently the slave axis(es). Slave(s) remain “engaged” in GEAR mode after the input-triggered halt.

SEE ALSO `GEAR, GEAR_OFF, GEAR_POS`

APPLICATION

See *DSPL Application Notes*

EXAMPLE

Axis 8 is the leader, axis 1is the follower with a gear ratio of 4.0. Axis 1 should “engage” at the occurrence of probe interrupt *EXT2.

```
GEAR_PROBE (0x8,0x1,2,4.0)
```

ICUBCOUNT

IDENTIFIER

IDENTIFIER Cubic Spline Table Index Counter

USAGE DSPL (PLC, Motion)

DESCRIPTION

`ICUBCOUNT` is a DSPL reserved word that is used to indicate to the DSPL program at which index the internal cubic spline (`CUBIC_INT`) is running.

SEE ALSO none

EXAMPLE

The DSPL line below checks the range of `ICUBCOUNT` as part of a conditional expression:

```
IF ((ICUBCOUNT > 1) AND (ICUBCOUNT < 5))
```

IF

FUNCTION IF Operand of IF-(then)-(ELSE)-ENDIF Structure

EXECUTION 200 microseconds

SYNTAX

```
IF (conditional expression)
    program code to execute if the IF condition is True
ELSE
    program code to execute if the IF condition is False
ENDIF
```

USAGE DSPL (PLC, Motion)

ARGUMENTS

conditional expression

The conditional expression must be boolean, equating to True or False. The conditional expression may consist of multiple boolean conditions ANDed or ORed together. The conditional expression operators are:

>	greater than
<	less than
>=	greater than or equal
<=	less than or equal
= =	equal
!=	not equal
AND	logical AND
OR	logical OR
&	bit-wise AND

The conditional expression is enclosed via sets of parentheses. Nested parentheses may be used when multiple boolean conditions are used or more complex conditional expressions are implemented.

IF cont.

Note: If nested parentheses are not used to indicate evaluation precedence in a conditional expression, the expression will be evaluated from left-to-right.

For example,

```
IF ( (VAR1 > 100) AND (POS2 > 100) AND
(ERR1 == 200) OR (IN_REG1 & 0x3) AND
(CVEL1 > 10 ) )
```

This line is interpreted in DSPL as:

```
IF ( { { { [ (VAR1 > 100) AND (POS2 > 100) ]
AND (ERR1 == 200) } OR
(IN_REG1 & 0x3) }AND ( (CVEL1 > 10) })
```

DESCRIPTION

The IF-(then)-ELSE structure is used for conditional program execution. When IF-(then)-ENDIF statements are used, Mx4 will test the boolean condition(s). The instruction(s) after the `IF` statement will be executed if the conditional expression is True, otherwise the instruction(s) after the `ENDIF` statement will be executed. If the complete IF-(then)-ELSE-ENDIF structure is used, the instruction(s) following the `ELSE` operand will be executed if the conditional expression evaluates to False, program flow will then continue to the next instruction following the `ENDIF` statement.

IF-(then)-(ELSE)-ENDIF structures may be nested.

SEE ALSO

`ELSE, ENDIF`

APPLICATION

See *Application Notes*

IF cont.

EXAMPLE

Bring the motion of axis three to a halt if VAR1 is equal to 0 and the following error of axis three is greater than 1000 counts. If the above condition is False, preset the position of axis one to 100000, and if VAR2 is equal to 1, preset the position of axis two to 2000 counts.

```
IF      ( (VAR1 == 0) AND (ERR3 > 1000) )
               STOP (0x4)
ELSE
        POS_PRESET (0x1,100000)
        IF (VAR2==1)
               POS_PRESET (0x2,2000)
        ENDIF
ENDIF
```

INDEX_POS1, …, INDEX_POS8 IDENTIFIER

IDENTIFIER Index Position State Variable

USAGE DSPL (PLC, Motion)

DESCRIPTION

An index position state variable holds a 32-bit two's complement integer value that represents the index position (in encoder edge counts) of the specified axis.

Name	Description
INDEX_POS1	axis 1 index position
INDEX_POS2	axis 2 index position
INDEX_POS3	axis 3 index position
INDEX_POS4	axis 4 index position
INDEX_POS8	axis 5 index position
INDEX_POS6	axis 6 index position
INDEX_POS7	axis 7 index position
INDEX_POS8	axis 8 index position

SEE ALSO `CPOS1, ERR1, POS1, PROBE_POS1`

EXAMPLE

The index position state variables can be used as follows:

- as one of the values used in conjunction with a DSPL arithmetic operation:

```
VAR1 = INDEX_POS2 + 1000
```

- as one of the arguments in a DSPL conditional expression:

```
WAIT_UNTIL(INDEX_POS3 >= VAR22)
```

INP1_REG, INP2_REG — IDENTIFIER

IDENTIFIER DSPL Input Registers 1 and 2.

USAGE DSPL (PLC, Motion)

DESCRIPTION

The real time status of [Mx4:22] [Mx4 Octavia:32] external user-defined inputs is available in DSPL in the 16-bit registers `INP1_REG` and `INP2_REG`. A set bit (bit = 1) indicates an active input condition.

The input bit registers may only be used with the bitwise operators in conditional expressions within the DSPL conditional structures, `IF`, `WHILE`, and `WAIT_UNTIL`. A user defined bit mask that must be used in conjunction with the bitwise operator & must follow the hexadecimal format 0x????, where ???? is a 16-bit hexadecimal mask. For example, a mask value of 0x0204 will mask out all bits except bits 2 and 9.

Name	Bit Format	Input
`inp1_reg`	bit 0	IN0
	bit 1	IN1
	bit 2	IN2
	bit 3	IN3
	bit 4	IN4
	bit 5	IN5
	bit 6	IN6
	bit 7	IN7
	bit 8	IN8
	bit 9	IN9
	bit 10	IN10
	bit 11	IN11
	bit 12	IN12
	bit 13	IN13
	bit 14	IN14
	bit 15	IN15

INP1_REG, INP2_REG cont. IDENTIFIER

Name	Bit Format	Input
inp2_reg	bit 0	IN16
	bit 1	IN17
	bit 2	IN18
	bit 3	IN19
	bit 4	IN20
	bit 5	IN21
	bit 6	IN22
	bit 7	IN23
	bit 8	IN24
	bit 9	IN25
	bit 10	IN26
	bit 11	IN27
	bit 12	IN28
	bit 13	IN29
	bit 14	IN30
	bit 15	IN31

SEE ALSO `~, &, AND, OR`

EXAMPLE

The conditional expression in the DSPL IF statement below will evaluate to TRUE if bit 0, 5, or 14 in input register 1 is set (bit = 1):

```
IF (INP1_REG & 0x4021)
```

INP_STATE

FUNCTION Configure Logic State of Inputs

EXECUTION 10 microseconds

SYNTAX `INP_STATE (inp1, inp2)`

USAGE DSPL (Motion), Host (command code: B4h)

ARGUMENTS

inp_1 bit coding the logic state of inputs

bit = 0	:	active LOW input
bit = 1	:	active HIGH input
bit 15	:	IN15
	.	
	.	
bit 0	:	IN0

inp_2 bit coding the logic state of inputs

bit = 0	:	active LOW input
bit = 1	:	active HIGH input
bit 15	:	IN31
	.	
	.	
bit 0	:	IN16

When used in DSPL, arguments inp_1 and inp_2 may be selected as variables.

INP_STATE cont.

DESCRIPTION

This command allows the user to define the logic state of the [Mx4:22] [Mx4 Octavia:32] inputs. Each input may be configured as active LOW or active HIGH (TTL logic levels) (the Mx4 inputs are level sensitive).

Note: At power-up and reset, Mx4 inputs default as active LOW.

SEE ALSO none

EXAMPLE

Configure the IN0 input as active HIGH input. The remaining inputs are to be configured as active LOW.

```
INP_STATE (0x0001,0x0000)
```

INPUT — Acc4 option command

FUNCTION read value from ASCII terminal

EXECUTION 200 microseconds

SYNTAX `INPUT (dvar)`

USAGE DSPL (Motion)

ARGUMENTS

dvar `VAR1-VAR128`. Specifies DSPL variable in which the value returned from the terminal is stored.

DESCRIPTION

The `INPUT` command is used to write a value sent by the ASCII terminal to the specified DSPL variable. The ASCII transmission to the terminal takes the format:

‘??’

The DSPL motion program from which the INPUT command was executed will halt (wait) program execution until the value is returned from the ASCII terminal. The ASCII transmission from the terminal to the Mx4 must follow the format:

Inp=x

Where x may range from =2147000000 <=x <= 2147000000. The value written is an integer with 3 implied fractional digits. For example inp=123456 will set the specified variable to 123.456.

EXAMPLE

Request ASCII input, assign to VAR15

```
INPUT (VAR15)
```

INT

FUNCTION Extract the Integer Portion of a Constant or a Variable Value.

EXECUTION 10 microseconds

SYNTAX `INT(valu)` *or* `-INT(valu)`

USAGE DSPL (PLC, Motion)

ARGUMENTS

valu A constant or a variable (VAR1 through VAR128)

DESCRIPTION

This function extracts the integer portion of a constant or a variable value. The integer portion of a number consists of all of the digits to the left of the decimal point. If a minus sign appears to the left of the INT function, the integer portion of *valu* is multiplied by -1.

Note: This function can only be used with a variable assignment statement. For example:

```
VAR55 = INT(VAR22)
```

SEE ALSO ABS, FRAC, SIGN, SQRT

EXAMPLE

The first example extracts the integer portion of the value stored in VAR64, and stores the negated result in VAR2:

```
VAR2 = -INT(VAR64)
```

The second example finds the integer portion of -61.839 and stores the result (-61) in VAR5:

```
VAR5 = INT(-61.839)
```

INT_HOST

FUNCTION Generate an Interrupt to the Host from DSPL

EXECUTION 10 microseconds

SYNTAX `INTHOST (id)`

USAGE DSPL (PLC, Motion)

ARGUMENTS

id interrupt signature or identifier

$0x00 \leq id \leq 0xFF$

DESCRIPTION

The `INT_HOST` command generates a hardware interrupt to the host upon its execution. The 8-bit identifier id will be copied to the Dual Port RAM at location 0x00E, and bit 4 in the interrupt register 2 (009h) will be set.

SEE ALSO none

APPLICATION

See *Application Notes*

EXAMPLE

Generate an interrupt to the host with an identifier byte equal to ABh.

```
INT_HOST (0xAB)
```

INT_REG_ALL_CLR

FUNCTION Clears the DSPL Interrupt and Input Bit Register Variables

EXECUTION 10 microseconds

SYNTAX `INT_REG_ALL_CLR ( )`

USAGE DSPL (PLC, Motion)

ARGUMENTS

none

DESCRIPTION

The `INT_REG_ALL_CLR` command clears the DSPL interrupt bit registers:

```
INDEX_REG          MOTCP_REG
ESTOP_REG          OFFSET_REG
FERR_REG           POSBRK_REG
FERR_REG           PROBE_REG
```

SEE ALSO `INT_REG_CLR`

APPLICATION

See *Application Notes*

EXAMPLE

Clear the DSPL Bit Register Variables.

```
INT_REG_ALL_CLR ( )
```

INT_REG_CLR

FUNCTION Clears the Specified DSPL Bit Register Variables

EXECUTION 10 microseconds

SYNTAX `INT_REG_CLR (m, mask1, ... , mask8)`

USAGE DSPL (PLC, Motion)

ARGUMENTS

m — bit coding specifying the interrupt registers to modify,

bit 9-15	:	unused
bit 8	:	ENCFLT_REG
bit 7	:	ESTOP_REG
bit 6	:	FERRH_REG
bit 5	:	FERR_REG
bit 4	:	OFFSET_REG
bit 3	:	PROBE_REG
bit 2	:	MOTCP_REG
bit 1	:	POSBRK_REG
bit 0	:	INDEX_REG

mask — a hexadecimal bit mask specifying which bits of the specified bit register are to be cleared. A set bit (bit=1) in the mask indicates the corresponding bit in the variable bit register is to be cleared.

DESCRIPTION

The `INT_REG_CLR` command is used to clear only the specified bits of selected variable bit register(s).

SEE ALSO `INT_REG_ALL_CLR`

INT_REG_CLR cont.

APPLICATION

See Application Notes

EXAMPLE

Clear the axis two and axis four following error interrupt bits of the `OFFSET_REG` bit register. Also, clear the `INDEX_REG` bits for all 4 axes.

```
INT_REG_CLR(0x0011,0xF,0xA0)
```

KILIMIT

FUNCTION Integral Gain Limit

EXECUTION 200 microseconds

SYNTAX `KILIMIT (n, val1, ... , val8)`

USAGE DSPL (Motion), Host (command code: 74h)

ARGUMENTS

n	bit coding of the specified axis(es)
val_x	value setting the limit of the integral action for each axis

Note: $0 \leq val \leq 14$

val = 0 indicates no limit on integration channels
val = 14 indicates maximum limit on integration channels

For example,

Kilimit val = 0	+/- 10v DAC action from K_i control law parameter
Kilimit val = 1	+/- 5v DAC action from K_i control law parameter
Kilimit val = 2	+/- 2.5v DAC action from K_i control law parameter
Kilimit val = 3	+/- 1.25v DAC action from K_i control law parameter
:	
:	

DESCRIPTION

This command is used to set the limit for integral action related to the choice of par_{x1} in the CTRL RTC. Integral limit is specified for each axis. Default val_x are set to zero (i.e., no limit on integration channels).

SEE ALSO `CTRL`

KILIMIT cont.

APPLICATION

This command clamps the integral channel by reducing this channel's saturation level. Reducing the saturation level will reduce the channel's depletion time. Using this instruction is essential where large integral gain is required. Clamping the integral channel will let the system zero position error without a lengthy "creeping motion" to its target position.

Command Sequence Example

```
CTRL ( )        ;set the gains
KILIMIT ( )     ;this instruction may be used before or after CTRL
```

EXAMPLE

Set a maximum limit on the integral action of axis 2, 3 and 4.

```
KILIMIT (0xE,14,14,14)
```

LINEAR_MOVE

FUNCTION Simple Constant Acceleration Linear Motion

EXECUTION 200 microseconds

SYNTAX `LINEAR_MOVE (n, pos1, vel1, ..., pos8, vel8)`

USAGE DSPL (Motion)

ARGUMENTS

n — bit coding of the specified axis(es)

pos_x — target position for axis x

$-2147483648 <= pos_x <= 2147483647$ counts

vel_x — target velocity for axis x

$-256 <= vel_x <= 255.99998$ counts/200µsec

When used in DSPL, arguments pos_x and vel_x may be selected as variables.

DESCRIPTION

The `LINEAR_MOVE` command allows the user to program a constant acceleration linear profile in any or all of the four axes. The user simply enters the target position and target velocity for the axis in question. The Mx4 will automatically calculate the required acceleration to accomplish the motion.

Upon execution of a `CIRCLE` or `LINEAR` related command, the DSPL program flow will proceed to the following command. If the following command is <u>not</u> a `CIRCLE` or `LINEAR` related command, it will be executed immediately. If the following command is a `CIRCLE` or `LINEAR` related command, it will be executed after the previous `CIRCLE`/ `LINEAR` motion is complete.

LINEAR_MOVE Cont.

Note: A `LINEAR_MOVE` command may not pass through the same position more than once. For example, a `LINEAR_MOVE` motion may not decelerate to zero velocity and continue decelerating (ie: change velocity polarity). If the above condition is violated, the `LINEAR_MOVE` motion will not be executed.

Note: The `LINEAR_MOVE` command will automatically calculate the acceleration for the motion. If the calculated acceleration is approximated to zero (ie: too small to be represented in the 16-bit fractional numerical range), the `LINEAR_MOVE` motion will not be executed.

SEE ALSO `CIRCLE, LINEAR_MOVE_S, LINEAR_MOVE_T`

APPLICATION

See *DSPL Application Notes*

EXAMPLE

From the present positions and velocities, move axes 1 and 4 to zero position with velocities of 1 and -2 counts/200µsec, respectively.

```
LINEAR_MOVE (0x9, 0, 1, 0, -2)
```

LINEAR_MOVE_S

FUNCTION Linear, S-Curve Motion

EXECUTION 200 microseconds

SYNTAX `LINEAR_MOVE_S (n, pi1, vi1, pt1, vt1, t1, a1, ... , pi8, vi8, pt8, vt8, t8, a8)`

USAGE DSPL (Motion)

ARGUMENTS

n	bit coding of the specified axis(es)
pi_x	initial starting position of axis x $-2147483648 \le pi_x \le 2147483647$ counts
vi_x	initial starting velocity of axis x $-256 \le vi_x \le 255.99998$ counts/200µs
pt_x	target position for axis x $-2147483648 \le pt_x \le 2147483647$ counts
vt_x	target velocity for axis x $-256 \le vt_x \le 255.99998$ counts/200µs
t_x	time for linear move motion to complete for axis x 0.1 ms $\le t_x \le 223$ minutes

LINEAR_MOVE_S cont.

Note: t_x has a default unit of 200µs, however the t_x value must be a multiple of 5ms. If t_x is not a multiple of 5ms, t_x will be truncated by the compiler.

a_x unsigned value specifying acceleration for linear move motion

$0 \le a_x \le 1.999969$ counts/(200µs)^2

DESCRIPTION

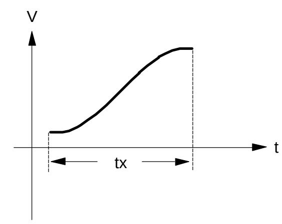
The `LINEAR_MOVE_S` command is a general purpose motion command that allows the user to accomplish S-Curve, constant acceleration, or constant velocity motion in any or all of the four axes.

Upon execution of a `CIRCLE` or `LINEAR` related command, the DSPL program flow will proceed to the following command. If the following command is not a `CIRCLE` or `LINEAR` related command, it will be executed immediately. If the following command is a `CIRCLE` or `LINEAR` related command, it will be executed after the previous `CIRCLE`/`LINEAR` motion is complete.

S-Curve Motion

The `LINEAR_MOVE_S` command can generate S-curve motion with the proper tx and a_x argument values.

LINEAR_MOVE_S cont.



S-Curve Velocity Profile

For S-curve motion, the t_x and a_x value must meet the following requirements:

$$t_x = \frac{2*(pt_x - pi_x)}{vt_x + vi_x}$$
$$a_x, min = |vt_x - vi_x| / t_x$$
$$a_x, max = 2*|vt_x - vi_x| / t_x$$
$$a_x, min \le a_x \le a_x, max$$

If the above t_x and a_x conditions are not met, the compiler will give a warning and recalculate the offending parameter(s).

Constant Acceleration Motion

A constant acceleration velocity profile may be achieved with the `LINEAR_MOVE_S` command by following these conditions:

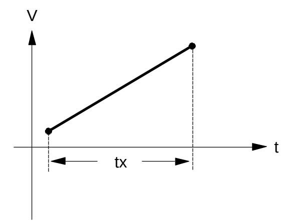
$t_x = 0$
$a_x = 0$
$vi_x \ne vt_x$

LINEAR_MOVE_S cont.

The compiler calculates the t_x and a_x values based on the P-V-T calculations,

$$t_x = \frac{2*(pt_x - pi_x)}{vt_x + vi_x}$$
$$pt_x = pi_x + (vt_x + vi_x)*t_x / 2$$
$$a_x = (vt_x - vi_x) / t_x$$

REMEMBER, t_x must evaluate to a multiplier of 5ms in the above equations.



Constant Acceleration Velocity Profile

Constant Velocity Motion

`LINEAR_MOVE_S` generates a constant velocity profile when the following conditions are met:

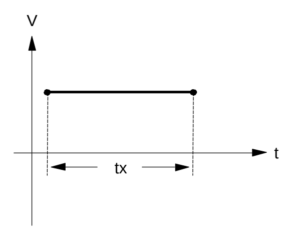
$t_x = 0$
$a_x = 0$
$vi_x = vt_x$

LINEAR_MOVE_S cont.

The compiler calculates the t_x value based on the P-V-T calculation,

$$pt_x = pi_x + vt_x * t_x$$
$$t_x = \frac{pt_x - pi_x}{vt_x}$$

Again, REMEMBER that t_x must evaluate to a multiple of 5ms in the above equation. Therefore, choose the P and V values accordingly.



Constant Velocity Profile

SEE ALSO `CIRCLE, LINEAR_MOVE, LINEAR_MOVE_T`

APPLICATION

See *Application Notes*

EXAMPLE 1 Constant Velocity

Move axis one from a current position of 50,000 counts to a target position of 100,000 counts with a constant velocity equal to 2.5 counts/200µs.

LINEAR_MOVE_S cont.

n	0x1
pi_1	50,000 counts
vi_1	2.5 counts /200µs
pt_1	100,000 counts
vt_1	2.5 counts /200µs
t_1	0 (200ms units)
a_1	0 counts / $(200\mu s)^2$

```
LINEAR_MOVE_S (0x1, 50000, 2.5, 100000, 2.5, 0, 0)
```

Note: The axis one velocity must equal 2.5 counts/200µs before executing the `LINEAR_MOVE_S` command ... remember vi_1 = 2.5.

EXAMPLE 2 Multi-Axis Motion

In addition to executing the axis one motion of Example 1, move axis three from an initial position, initial velocity (0,0) to target position, target velocity (10000, 5.0) with constant acceleration.

n	0x5
pi_1	50,000 counts
vi_1	2.5 counts /200µs
pt_1	100,000 counts
vt_1	2.5 counts /200µs
t_1	0 (200ms units)
a_1	0 counts / $(200\mu s)^2$
pi_3	0 counts

LINEAR_MOVE_S cont.

vi_3	0 counts /200µs
pt_3	10,000 counts
vt_3	5.0 counts /200µs
t_3	0 (200ms units)
a_3	0 counts / $(200\mu s)^2$

```
LINEAR_MOVE_S (0x5, 50000, 2.5, 100000, 2.5, 0, 0, 0, 0, 10000,
5.0, 0, 0)
```

EXAMPLE 3　S-Curve Motion

Move axis four from initial position, initial velocity (1000, 1.0) to target position, target velocity (11000, 4.0) with S-curve velocity profile utilizing minimum acceleration.

n	0x8
pi_4	1,000 counts
vi_4	1.0 counts /200µs
pt_4	11,000 counts
vt_4	4.0 counts /200µs

t_4

$$\frac{2(11,000-1,000)}{4.0+1.0} = 4,000 \text{ (200}ms\text{ units)}$$

a_4

$$a_4\text{, min} = \left|4.0-1.0\right| / 4,000 = 0.0075 \text{ counts / } (200ms)^2$$

```
LINEAR_MOVE_S(0x8,    1000,    1.0,    11000,    4.0,
            4000,0.00075)
```

LINEAR_MOVE_T

FUNCTION Simple Time-Based Constant Acceleration Linear Motion

EXECUTION 200 microseconds

SYNTAX `LINEAR_MOVE_T (n, pos`$_1$`, tm`$_1$`, ..., pos`$_8$`, tm`$_8$`)`

USAGE DSPL (Motion)

ARGUMENTS

pos_x target position for axis x

-2147483648 <= pos_x <= 2147483647 counts

tm_x motion time for axis x

$0 \leq tm_x \leq 5000000$ (200μs)

Note: The time argument, tm_x, is an unsigned value with a unit of 200usec.

When used in DSPL, arguments pos_x and tm_x may be selected as variables.

DESCRIPTION

The `LINEAR_MOVE_T` command allows the user to program a constant acceleration linear profile in any or all of the four axes. The user simply enters the target position and time to complete the move for the axis in question, and the Mx4 will automatically calculate the required acceleration and velocity to accomplish the motion.

Upon execution of a `CIRCLE` or `LINEAR` related command, the DSPL program flow will proceed to the following command. If the following command is <u>not</u> a `CIRCLE` or `LINEAR` related command, it will be executed immediately. If the following command is a `CIRCLE` or `LINEAR` related command, it will be executed after the previous `CIRCLE`/ `LINEAR` motion is complete.

LINEAR_MOVE_T cont.

Note: A `LINEAR_MOVE_T` command may not pass through the same position more than once. For example, a `LINEAR_MOVE_T` motion may not decelerate to zero velocity and continue decelerating (ie: change velocity polarity). If the above condition is violated, the `LINEAR_MOVE_T` motion will not be executed.

Note: The `LINEAR_MOVE_T` command will automatically calculate the acceleration for the motion. If the calculated acceleration is approximated to zero (ie: too small to be represented in the 16-bit fractional numerical range), the `LINEAR_MOVE_T` motion will not be executed.

SEE ALSO `CIRCLE, LINEAR_MOVE, LINEAR_MOVE_S`

APPLICATION

See *Application Notes*

EXAMPLE

From the present positions and velocities, move axes 1 and 4 to zero position in 1.5 seconds.

```
LINEAR_MOVE_T (0x9, 0, 7500, 0, 7500)
```

LOW_PASS (option)

FUNCTION	Implement Low Pass Filter at Controller Output
EXECUTION	200 microseconds
SYNTAX	LOW_PASS (n, $freq_x$)
USAGE	DSPL (Motion), Host (command code: 8Eh*)

Note: This RTC code (8Eh) is the same as the one used with NOTCH, therefore one option (either LOW_PASS or NOTCH) can be used at any time.

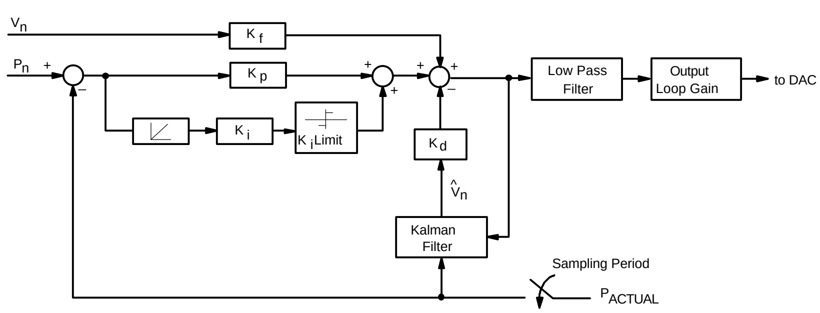
ARGUMENTS

n	bit coding of the only specified axis
$freq_x$	unsigned value specifying the low pass filter cut-off frequency for axis x
	$0 \le freq_x \le 1850$

When used in DSPL, the argument $freq_x$ may be selected as a variable.

DESCRIPTION

This command implements a low pass filter at the controller output for the specified axis.



Mx4 Block Diagram with Low Pass Filter

LOW_PASS cont.

The low pass filter implements the following transfer function:

$$G(s) = \frac{w_n^2}{s^2 + 2zw_n \cdot s + w_n^2}$$

where, $w_n = 2pf_n$, f_n = cut-off frequency, and $z = 0.6$

The frequency and bandwidth of the low pass filter is programmable.

Note: By programming a cut-off frequency of 0, the low pass filter for the specified axis is disabled.

SEE ALSO none

EXAMPLE: DSPL Programming Low Pass

1) Set a low pass filter at 250 Hz for axis 2 (see below).

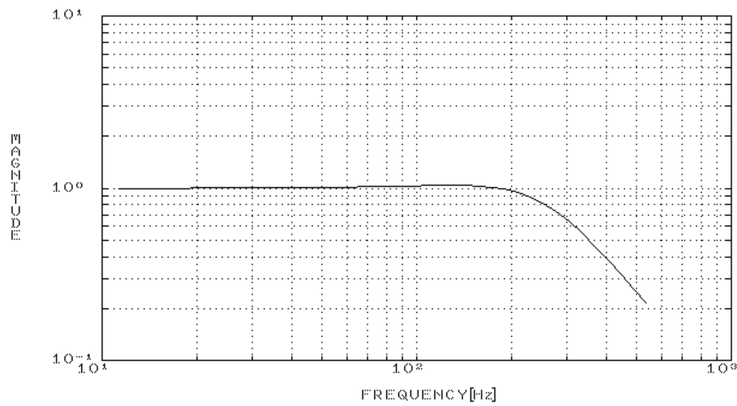
```
LOW_PASS (0x2,250)
```

2) Disable the low pass filter of axis 1.

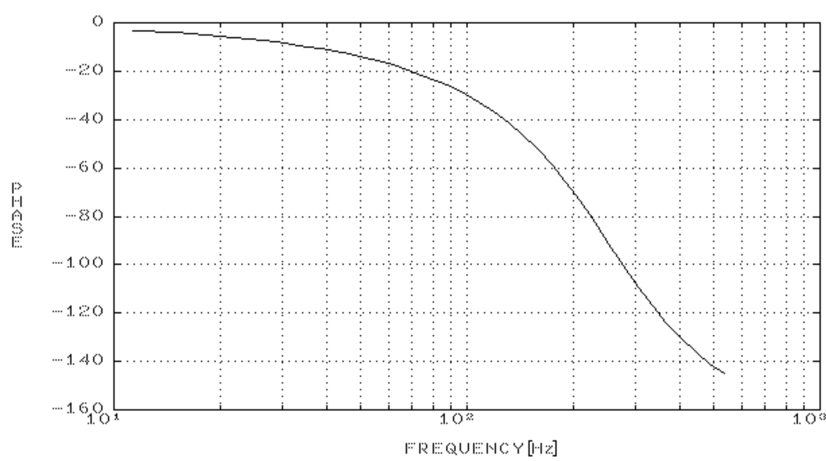
```
LOW_PASS (0x1,0)
```

Note: Mx4 default setting for low pass filter is no filter (or filter disabled.

LOW_PASS cont.



Magnitude Diagram



Phase Diagram of 250 Hz Low Pass Filter

MAXACC

FUNCTION Maximum Acceleration

EXECUTION 100 microseconds

SYNTAX `MAXACC (n, acc1, ... , acc8)`

USAGE DSPL (Motion), Host (command code: 71h)

ARGUMENTS

n	bit coding of the specified axis(es)
acc_x	unsigned value specifying the maximum acceleration / deceleration for axis x

$0 \le acc_x \le 1.999969$ counts/(200µs)^2

Note: Acceleration is partitioned into 1 bit integer, 15 bits fraction.

When used in DSPL, argument `accx` may be selected as a variable.

DESCRIPTION

This command specifies the maximum acceleration / deceleration for the axes specified. The maximum acceleration values are used in the `STOP` and `VELMODE` commands.

Note: `MAXACC` will be ignored if the specified argument is zero.

SEE ALSO `ESTOP_ACC, STOP, VELMODE`

MAXACC cont.

APPLICATION

This command sets the maximum acceleration affordable by the servo drive and motor combination. It is useful to program this parameter such that the system will not go to control saturation during `VELMODE` or `STOP`.

Command Sequence Example

```
MAXACC ( )      ;set the maximum accel. so system can be stopped
CTRL ( )        ;set the gains
KILIMIT ( )
.
.
AXMOVE ( )      ;run system in axis move
VELMODE ( )     ;run system in velocity mode
```

EXAMPLE

Set a maximum acceleration for axes 2 and 3 of 0.25 encoder counts / $(200\mu s)^2$.

```
MAXACC (0x6,0.25,0.25)
```

MOTOR_PAR

Vx4++ option command

FUNCTION Motor Parameter

EXECUTION 200 microseconds

SYNTAX `MOTOR_PAR (n, mpar`$_1$`, ... , mpar`$_8$`)`

USAGE DSPL (Motion), Host (command code: 76h)

ARGUMENTS

n — bit coding of the specified axis(es)

$mpar_x$ — for AC induction motor, defines the motor slip gain

$-32768 \le fval_x \le 32767$

for brushless DC motor, defines the commutation angle

$-32768 \le fval_x \le 32767$

When used in DSPL, the argument $mpar_x$ may be selected as a variable.

DESCRIPTION

The `MOTOR_PAR` command defines motor technology-dependent parameters. If the axis in question is an AC induction motor, the command defines the motor slip gain. If the axis is a brushless DC motor, the command defines the commutation angle (in encoder counts).

Note: The `MOTOR_PAR` command does not need to be programmed for brushtype DC motors.

SEE ALSO none

APPLICATION

See *Vx4++ User's Guide*

EXAMPLE

Program a slip gain equal to 5500 for axes two, three, and four (the motors are identical AC induction motors)

```
MOTOR_PAR (0xE, 5500, 5500, 5500)
```

MOTOR_TECH

Vx4++ option command

FUNCTION Motor Technology

EXECUTION 200 microseconds

SYNTAX `MOTOR_TECH (n, mtech`$_1$`, ... , mtech`$_8$`)`

USAGE DSPL (Motion), Host (command code: 7Ch)

ARGUMENTS

n	bit coding of the specified axis(es)
$mtech_x$	for AC induction, $mtech_x$ = `AC_IND`
	for brushless DC, $mtech_x$ = `BRUSHLESS_DC`
	for brushtype DC, $mtech_x$ = `DC`

DESCRIPTION

Mx4 with the Vx4++ drive control option is capable of controlling brushtype DC, AC induction, and brushless DC motors. This command allows the motor technology of each axis to be programmed.

Note: Mx4 with Vx4++ will not execute the `MOTOR_TECH` command if the `Vx4_BLOCK` command is active for the axes in question.

SEE ALSO `Vx4_BLOCK`

APPLICATION

See *Vx4++ User's Guide*

EXAMPLE

Select brushless DC technology for axis one, brushtype DC for axis two, and AC induction technology for axis four.

```
MOTOR_TECH (0xB, BRUSHLESS_DC, DC, AC_IND)
```

NOTCH (option)

FUNCTION Implement Notch Filter at Controller Output

EXECUTION 200 microseconds

SYNTAX `NOTCH (n, freq`$_X$`, q`$_X$`)`

USAGE DSPL (Motion), Host (command code: 8Eh*)

Note: This RTC code (8Eh) is the same as the one used with `LOW_PASS`, therefore one option (either `NOTCH` or `LOW_PASS`) can be used at any time.

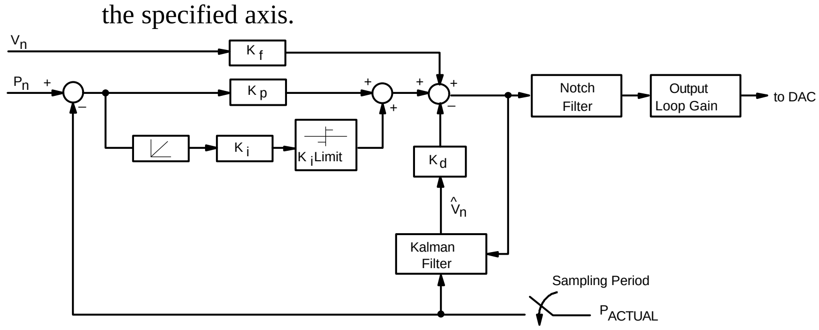
ARGUMENTS

n	bit coding of the only specified axis
$freq_x$	unsigned value specifying the notch filter frequency for axis x $0 \le freq_x \le 1650$ Hz
q_x	unsigned value specifying the notch filter quality factor for axis x $q_x = 1$ ~25% bandwidth filter $q_x = 2$ ~10% bandwidth filter

When used in DSPL, the arguments $freq_x$ and q_x may be selected as variables.

DESCRIPTION

This command implements a notch filter at the controller output for the specified axis.



Mx4 Block Diagram with Notch Filter

NOTCH cont.

The notch filter implements the transfer function:

$$G(s) = \frac{s^2 + w_n^2}{s^2 + \frac{w_n}{Q} s + w_n^2}$$

where, $w_n = 2 p f_n$ and f_n = notch frequency

The frequency and bandwidth of the notch is programmable.

Note: By programming a notch frequency of 0, the notch filter for the specified axis is disabled.

SEE ALSO none

EXAMPLE: DSPL Programming Notch

1) Set a notch filter at 750 Hz with a narrow bandwidth (q = 2) for axis 2 (see Fig. 4-3 below).

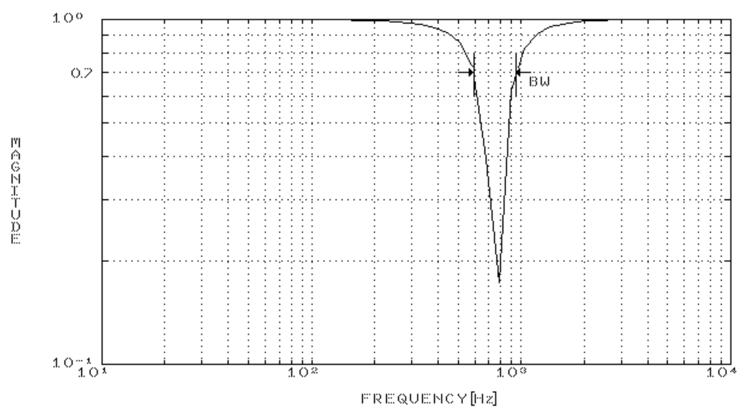
```
NOTCH (0x2,750,2)
```

2) Disable the notch filter of axis 1.

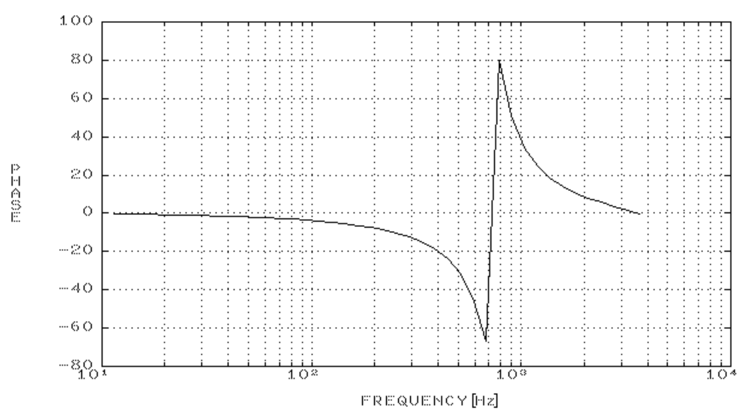
```
NOTCH (0x1,0,1)
```

Note: The Mx4 default setting for notch filter is no notch (or notch disabled.

NOTCH cont.



(a)



(b)

Frequency Response of Discrete 750 Hz, Q=2 Notch Filter

OFFSET

FUNCTION Amplifier Offset Cancellation

EXECUTION 200 microseconds

SYNTAX `OFFSET (n)`

USAGE DSPL (Motion), Host (command code: 5Fh)

ARGUMENTS

n bit coding the ONLY axis involved

DESCRIPTION

This command minimizes the offset generated by the D/A Converter (DAC). Upon completion of offset tuning, an interrupt is generated to the host. The condition is recorded in DPR interrupt status register location 009h. DPR status register location 00Ch will identify the axis responsible. Bit 6 of DPR locations [Mx4:7FEh] [Mx4 Octavia:1FFEh] is also set.

The interrupt condition is also axis bit-coded in bits 0-3 of the DSPL `OFFSET_REG` bit register.

Note: `OFFSET` may be run with only one axis at a time. The status of the remaining three axes is not affected by running `OFFSET`.

To run `OFFSET`, the following steps should be followed for the corresponding axis:

1. The axis should be in closed loop with optimal gains set.
2. K_i must be non zero for the axis.
3. The axis should be 'stopped', with no motion commands in progress.
4. Start `OFFSET` with the specified axis.
5. Offset adjust is complete when a host interrupt is generated.

SEE ALSO `CTRL`

OFFSET cont.

APPLICATION

Most servo amplifiers on the market present an input offset voltage problem that is undesirable for an accurate positioning application. Using `OFFSET` you may neutralize amplifier offset. To make this happen, you must:

1. enable `OFFSET` for the axis whose offset is to be neutralized, and
2. use a non-zero K_i gain that maintains stability and zeros position error. (It is assumed that other control gains are selected such that the system is stable.)

Position error is integrated via the integral channel until position error is forced to zero. In the absence of amplifier offset, the DAC voltage that would have achieved zero position error is zero. Any non-zero DAC value is due to an error caused by amplifier offset voltage. Mx4 measures the voltage, reports satisfactory completion of the `OFFSET` command (generates an interrupt) and uses this measured voltage value to neutralize offset throughout the entire control operation (until machine is turned off). Due to the variable nature of amplifier offset, offset calibration may be necessary any time the machine is turned on.

Command Sequence Example

```
MAXACC ( )       ;set the maximum accel. so system can be stopped
CTRL ( )         ;set the gains
KILIMIT ( )      ;put system in a position loop, make sure integral
                 ;gain is non-zero
.
.
OFFSET ( )
```

EXAMPLE

After verifying that `OFFSET` Steps 1-3 (see DESCRIPTION, above) have been followed, do offset tuning for axis 3.

```
OFFSET (0x4)
```

OUTGAIN

FUNCTION Output Loop Gain

EXECUTION 200 microseconds

SYNTAX `OUTGAIN (n, m1, ... , m8)`

USAGE DSPL (Motion), Host (command code: 81h)

ARGUMENTS

n	bit coding of the specified axis(es)
m_x	value which defines the output gain for axis x

m=0	gain=1
m=1	gain=2
m=2	gain=4
m=3	gain=8
m=4	gain=16

When used in DSPL, argument m_x may be selected as a variable.

DESCRIPTION

This command is used to set the gain for the output of the position loops. The default m is set to zero (gain = 1).

Note: Please see block diagram with `CTRL` command.

SEE ALSO `CTRL`

APPLICATION

In applications where the number of position encoder counts (per mechanical revolution of the shaft) is low, lack of resolution in the feedback path will manifest itself as a low gain. This may be compensated for by a loop gain adjustment. In practice, this command may use an argument greater than 1 if the encoder line number is less than 1000.

OUTGAIN cont.

Command Sequence Example

```
MAXACC ( )       ;set the maximum accel. so system can be stopped
CTRL ( )         ;set the gains
KILIMIT ( )
OUTGAIN ( )
```

EXAMPLE

Program output loop gains of eight for axis 3 and two for axis 4.

```
OUTGAIN (0xC,3,1)
```

OUTP_OFF

FUNCTION Set Outputs to 'Off' State

EXECUTION 25 microseconds

SYNTAX [Mx4]

`OUTP_OFF (outp`$_1$`)`

[Mx4 Octavia]

`OUTP_OFF (outp`$_1$`, outp`$_2$`)`

USAGE DSPL (PLC, Motion), Host (command code: 55h)

ARGUMENTS

$outp_1$ bit coding of the outputs

if bit=0	no change in output status
if bit=1	output = HIGH TTL voltage

bit 15	OUT15 output
bit 14	OUT14 output
bit 13	OUT13 output
bit 12	OUT12 output
bit 11	OUT11 output
bit 10	OUT10 output
bit 9	OUT9 output
bit 8	OUT8 output
bit 7	OUT7 output
bit 6	OUT6 output
bit 5	OUT5 output
bit 4	OUT4 output
bit 3	OUT3 output
bit 2	OUT2 output
bit 1	OUT1 output
bit 0	OUT0 output

OUTP_OFF cont.

$outp_2$ bit coding of the outputs

if bit=0	no change in output status
if bit=1	output = HIGH TTL voltage

bit 15	OUT31 output
bit 14	OUT30 output
bit 13	OUT29 output
bit 12	OUT28 output
bit 11	OUT27 output
bit 10	OUT26 output
bit 9	OUT25 output
bit 8	OUT24 output
bit 7	OUT23 output
bit 6	OUT22 output
bit 5	OUT21 output
bit 4	OUT20 output
bit 3	OUT19 output
bit 2	OUT18 output
bit 1	OUT17 output
bit 0	OUT16 output

When used in DSPL, arguments $outp_1$ and $outp_2$ may be selected as variables.

DESCRIPTION

This command allows the 'OFF' status of all [Mx4:13] [Mx4 Octavia:32]outputs to be set.

SEE ALSO

`OUTP_ON, POSBRK_OUT`

APPLICATION

This command can be used for a general purpose logical output operation.

EXAMPLE

Turn 'off' the OUT0, OUT5, OUT6, and OUT12 outputs.

```
OUTP_OFF (0x1061, 0x0000)
```

OUTP_ON

FUNCTION Set Outputs to 'On' State

EXECUTION 25 microseconds

SYNTAX [Mx4]

`OUTP_ON (outp1)`

[Mx4 Octavia]

`OUTP_ON (outp1, outp2)`

USAGE DSPL (PLC, Motion), Host (command code: 56h)

ARGUMENTS

$outp_1$ bit coding of the outputs

if bit=0	no change in output status
if bit=1	output = LOW TTL voltage

bit 15	OUT15 output
bit 14	OUT14 output
bit 13	OUT13 output
bit 12	OUT12 output
bit 11	OUT11 output
bit 10	OUT10 output
bit 9	OUT9 output
bit 8	OUT8 output
bit 7	OUT7 output
bit 6	OUT6 output
bit 5	OUT5 output
bit 4	OUT4 output
bit 3	OUT3 output
bit 2	OUT2 output
bit 1	OUT1 output
bit 0	OUT0 output

OUTP_ON cont.

$outp_2$	bit coding of the outputs

if bit=0	no change in output status
if bit=1	output = LOW TTL voltage

bit 15	OUT31 output
bit 14	OUT30 output
bit 13	OUT29 output
bit 12	OUT28 output
bit 11	OUT27 output
bit 10	OUT26 output
bit 9	OUT25 output
bit 8	OUT24 output
bit 7	OUT23 output
bit 6	OUT22 output
bit 5	OUT21 output
bit 4	OUT20 output
bit 3	OUT19 output
bit 2	OUT18 output
bit 1	OUT17 output
bit 0	OUT16 output

When used in DSPL, arguments $outp_1$ and $outp_2$ may be selected as variables.

DESCRIPTION

This command allows the 'ON' status of all [Mx4:13] [Mx4 Octavia:32] outputs to be set.

SEE ALSO

`OUTP_OFF, POSBRK_OUT`

APPLICATION

This command can be used for a general purpose logical output operation.

EXAMPLE

Enable or turn 'on' the OUT1, OUT11, and OUT12 outputs.

```
OUTP_ON (0x1802, 0x0000)
```

OVERRIDE

FUNCTION Feedrate override for `CIRCLE/LINEAR`

EXECUTION 10 microseconds

SYNTAX `OVERRIDE (Val)`

USAGE DSPL (PLC, Motion), Host (command code:8Bh)

ARGUMENTS

`Val` Feedrate override multiplier

$0.1 \leq$ `Val` ≤ 10

When used in DSPL, argument `Val` may be selected as a variable.

DESCRIPTION

This command is used to set the feedrate override for the `CIRCLE` and `LINEAR` related commands.

SEE ALSO `CIRCLE, LINEAR_MOVE, LINEAR_MOVE_S, LINEAR_MOVE_T`

APPLICATION

none

EXAMPLES

Set a feedrate override of 4x.

```
OVERRIDE (4.0)
```

PI

IDENTIFIER

IDENTIFIER DSPL Constant representing π

USAGE DSPL (PLC, Motion)

DESCRIPTION

The identifier PI is a DSPL reserved word that provides a floating point approximation to the value π (3.14159265).

EXAMPLES

The identifier PI can be used as follows:

- to replace constant values in arithmetic expressions:

```
VAR3 = PI
VAR4 = 2 * PI
VAR9 = PI - 2
```

- to specify the value of an argument in a DSPL function:

```
VAR1 = SIN(PI)
```

- to replace a constant value in a conditional expression:

```
WAIT_UNTIL(VAR12 > PI)
```

POS1, …, POS8 — IDENTIFIER

IDENTIFIER Actual Position State Variable

USAGE DSPL (PLC, Motion)

DESCRIPTION

An actual position state variable holds a 32-bit two's complement integer value that represents the current position (in encoder edge counts) of the specified axis.

Name	Description
`POS1`	axis 1 actual position
`POS2`	axis 2 actual position
`POSx`	axis x actual position
.	
.	
`POS8`	axis 8 actual position

SEE ALSO `CPOS1, ERR1, INDEX_POS1, PROBE_POS1`

EXAMPLE

The actual position state variables can be used as follows:

- as one of the values used in conjunction with a DSPL arithmetic operation:

```
VAR1 = POS2 + VAR3
```

- as one of the arguments in a DSPL conditional expression:

```
IF(POS1 <= VAR2)
```

POSBRK_OUT

FUNCTION Set Outputs After Position Breakpoint Interrupt

EXECUTION 50 microseconds

SYNTAX [Mx4]

`POSBRK_OUT (n, outpon`$_{n1}$`, outpoff`$_{n1}$`, ... )`

[Mx4 Octavia]

`POSBRK_OUT (n, outpon`$_{n1}$`, outpon`$_{n2}$`, outpoff`$_{n1}$`, outpoff`$_{n2}$`, ... )`

USAGE DSPL (Motion), Host (command code: 7Dh)

ARGUMENTS

n	bit coding of the specified axis(es)
outpon$_{x1}$	bit coding the outputs to turn 'on' upon occurrence of position breakpoint interrupt (`EN_POSBRK`) for axis x.

if bit=0	no change in output status
if bit=1	output = LOW TTL voltage

bit 15	OUT15 output
bit 14	OUT14 output
bit 13	OUT13 output
bit 12	OUT12 output
bit 11	OUT11 output
bit 10	OUT10 output
bit 9	OUT9 output
bit 8	OUT8 output
bit 7	OUT7 output
bit 6	OUT6 output
bit 5	OUT5 output
bit 4	OUT4 output
bit 3	OUT3 output
bit 2	OUT2 output
bit 1	OUT1 output
bit 0	OUT0 output

POSBRK_OUT cont.

$outpon_{x2}$	bit coding the outputs to turn 'on' upon occurrence of position breakpoint interrupt (`EN_POSBRK`) for axis x.

if bit=0	no change in output status
if bit=1	output = LOW TTL voltage

bit 15	OUT31 output
bit 14	OUT30 output
bit 13	OUT29 output
bit 12	OUT28 output
bit 11	OUT27 output
bit 10	OUT26 output
bit 9	OUT25 output
bit 8	OUT24 output
bit 7	OUT23 output
bit 6	OUT22 output
bit 5	OUT21 output
bit 4	OUT20 output
bit 3	OUT19 output
bit 2	OUT18 output
bit 1	OUT17 output
bit 0	OUT16 output

POSBRK_OUT cont.

$outpoff_{x1}$	bit coding the outputs to turn ‘off’ upon occurrence of position breakpoint interrupt (`EN_POSBRK`) for axis x.

if bit=0	no change in output status
if bit=1	output = HIGH TTL voltage

bit 15	OUT15 output
bit 14	OUT14 output
bit 13	OUT13 output
bit 12	OUT12 output
bit 11	OUT11 output
bit 10	OUT10 output
bit 9	OUT9 output
bit 8	OUT8 output
bit 7	OUT7 output
bit 6	OUT6 output
bit 5	OUT5 output
bit 4	OUT4 output
bit 3	OUT3 output
bit 2	OUT2 output
bit 1	OUT1 output
bit 0	OUT0 output

POSBRK_OUT cont.

$outpoff_{x2}$	bit coding the outputs to turn 'off' upon occurrence of position breakpoint interrupt (EN_POSBRK) for axis x.

if bit=0	no change in output status
if bit=1	output = HIGH TTL voltage

bit 15	OUT31 output
bit 14	OUT30 output
bit 13	OUT29 output
bit 12	OUT28 output
bit 11	OUT27 output
bit 10	OUT26 output
bit 9	OUT25 output
bit 8	OUT24 output
bit 7	OUT23 output
bit 6	OUT22 output
bit 5	OUT21 output
bit 4	OUT20 output
bit 3	OUT19 output
bit 2	OUT18 output
bit 1	OUT17 output
bit 0	OUT16 output

When used in DSPL, arguments outpon and outpoff may be selected as variables.

DESCRIPTION

This command enables the output status of selected outputs to be activated by the occurrence of a position breakpoint interrupt (EN_POSBRK) for a specified axis. The POSBRK_OUT need only be executed once (ie: during initialization) unless the on/off output status desired changes. The specified outputs will change state as programmed through the $outpon_x$ and $outpoff_x$ arguments when an axis (axis x) generates a position breakpoint interrupt. The position breakpoint interrupt (EN_POSBRK) must be enabled for the output status changes to occur.

POSBRK_OUT cont.

SEE ALSO `EN_POSBRK, OUTP_OFF, OUTP_ON`

APPLICATION

This command can be used for an output operation where the output status must be tightly coupled to the position of one or more axes.

Command Sequence Example

```
EN_POSBRK       ;enable the pos breakpoint int for specified axis(es)
POSBRK_OUT      ;set the desired output status changes
```

EXAMPLE

If a position breakpoint interrupt occurs on axis 1, turn on OUT0-OUT3 and turn off OUT4.

```
POSBRK_OUT (0x1, 0x000F, 0x0000, 0x0010, 0x0000)
```

POS_PRESET

FUNCTION Preset Position Counter

EXECUTION 200 microseconds

SYNTAX `POS_PRESET (n, pset`$_1$`, ... , pset`$_8$`)`

USAGE DSPL (Motion), Host (command code 68h)

ARGUMENTS

n	bit coding of the specified axis(es)
$pset_x$	position counter preset value for axis x
	$-2147483648 \leq pset_x \leq 2147483647$ counts

When used in DSPL, argument $pset_x$ may be selected as a variable.

DESCRIPTION

This command will define the present position point for the axes specified.

Note: `POS_PRESET` will automatically disable the position breakpoint interrupt (if enabled). `POS_PRESET` should be executed only when the axes specified are not in motion.

SEE ALSO `POS_SHIFT`, `EN_POSBRK`

APPLICATION

This command is useful when the position counter must be forced to a new value. `POS_PRESET` may be used in the establishment of a new reference position. Please also see `POS_SHIFT`.

Command Sequence Example

No preparation is required before running this instruction.

EXAMPLE

Preset the axis 1 and axis 4 positions to 20000 and -45999 counts, respectively.

```
POS_PRESET (0x9,20000,-45999)
```

POS_SHIFT

FUNCTION Position Reference Shift

EXECUTION 200 microseconds

SYNTAX `POS_SHIFT (n, psft1, ... , psft8)`

USAGE DSPL (Motion), Host (command code: 5Dh)

ARGUMENTS

n	bit coding of the specified axis(es)
$psft_x$	position reference value for axis x
	$-2147483648 \le psft_x \le 2147483647$

When used in DSPL, the argument $psft_x$ may be selected as a variable.

DESCRIPTION

This command will shift the present position for the axes specified.

Note: `POS_SHIFT` will automatically disable the position breakpoint interrupt (if enabled) of the specified axes.

SEE ALSO `POS_PRESET, EN_POSBRK`

APPLICATION

This command may be used in homing a linear system based on index pulse position recording. Adding offset position (in encoder edge counts) to an already recorded position, presets position to a new value without losing position integrity (i.e., no counter information is lost). See also `EN_INDEX` and `POS_PRESET`.

Command Sequence Example

No preparation is required before running this instruction.

EXAMPLE

The current axis one position is 45000 counts. Shift the axis 1 position to 50000 counts. The current axis 3 position is 55000 counts. Shift the axis 3 position to 50000 counts.

```
POS_SHIFT (0x5,5000,-5000)
```

PRINT

Acc4 option command

FUNCTION Write (send) value to terminal

EXECUTION 200 microseconds

SYNTAX `PRINT` (value)

USAGE DSPL (Motion)

ARGUMENTS

value	32-bit two's complement constant or (integer) contents of specified DSPL variable.

DESCRIPTION

The `PRINT` command is used to write (send) a value to the ASCII terminal display. The ASCII transmission to the terminal takes the format:

(value) + <CR> + <LF> + '>'

The value displayed is an integer with 3 implied fractional digits. For example, 123456 is the value 123.456.

EXAMPLE

Write the value 100.45 to the ASCII terminal.

```
PRINT (100450)
```

Write the value contained in DSPL variable `VAR62` to the ASCII terminal.

```
PRINT (VAR62)
```

PRINTS

Acc4 option command

FUNCTION Write (send) ASCII String to Terminal

EXECUTION 200 microseconds

SYNTAX `PRINTS (“string”)`

USAGE DSPL (Motion)

ARGUMENTS

string — character string up to 26 characters in lengths. The string must consist of the printable ASCII characters (32-126).

DESCRIPTION

The `PRINT` command is used to write (send) a character string to the ASCII transmission to the terminal takes the format:

(string) + <CR> + <LF> + ’>’

EXAMPLE

Write “hello world” to the ASCII terminal.

`PRINT` (“hello world”)

PROBE_POS1, …, PROBE_POS8 IDENTIFIER

IDENTIFIER Probe Position State Variable

USAGE DSPL (PLC, Motion)

DESCRIPTION

A probe position state variable holds a 32-bit two's complement integer value that represents the probe position (in encoder edge counts) of the specified axis.

Name	Description
`PROBE_POS1`	axis 1 probe position
`PROBE_POS2`	axis 2 probe position
`PROBE_POSx`	axis x probe position
.	
.	
`PROBE_POS8`	axis 8 probe position

SEE ALSO `CPOS1, ERR1, INDEX_POS1, POS1`

EXAMPLE

The probe position state variables can be used as follows:

- as one of the values used in conjunction with a DSPL arithmetic operation:

```
VAR1 = PROBE_POS2 + 1000
```

- as one of the arguments in a DSPL conditional expression:

```
WHILE(PROBE_POS4 > VAR42)
```

PWM_FREQ

Vx4++ option command

FUNCTION Set Pulse Width Modulation (PWM) Frequency

EXECUTION 200 microseconds

SYNTAX `PWM_FREQ (m, pwm`$_1$`, pwm`$_2$`)`

USAGE DSPL (Motion), Host (command code: 7Fh)

ARGUMENTS

m — bit coding of the specified axis groups

m = 0x3	set axes one, two PWM frequency
m = 0xC	set axes three, four PWM frequency
m = 0xF	set axes one, two, three, four PWM frequency

pwm_1 — PWM frequency for axes one, two
pwm_2 — PWM frequency for axes three, four

$1.0 \leq pwm_x \leq 31.0$ kHz

DESCRIPTION

The frequency of the Vx4++ pulse width modulation outputs may be programmed via the `PWM_FREQ` command. The outputs may be programmed in axis pairs.

Note: Mx4 with Vx4++ will not execute the `PWM_FREQ` command if the `Vx4_BLOCK` command is active for the axes in question.

SEE ALSO `Vx4_BLOCK`

APPLICATION

See *Vx4++ User's Guide*

EXAMPLE

Set a PWM frequency of 15.4 kHz for axes three and four.

```
PWM_FREQ (0xC, 15.4)
```

REL_AXMOVE

FUNCTION Relative Position Axis Move with Trapezoidal Trajectory

EXECUTION 200 microseconds

SYNTAX

```
REL_AXMOVE (n, acc1, pos1, vel1, ... , acc8, pos8,
vel8)
```

USAGE DSPL (Motion), Host (command code: B7h)

ARGUMENTS

n — bit coding of the specified axis(es)

acc_x — unsigned value specifying the maximum halting acceleration (deceleration) for axis x

$0 \le acc_x \le 1.999969$ counts/(200µs)^2

pos_x — incremental position for axis x

$-805306367 \le pos_x \le 805306367$ counts

vel_x — unsigned target velocity for axis x

$0 \le vel_x \le 255.99998$ counts/200µs

When used in DSPL, arguments acc_x, pos_x and vel_x may be selected as a variable.

DESCRIPTION

The `REL_AXMOVE` command is similar to the `AXMOVE` command with the exception that relative (or incremental) position is specified, rather than an end position as with `AXMOVE`.

SEE ALSO `AXMOVE, AXMOVE_S, AXMOVE_T, REL_AXMOVE_S, REL_AXMOVE_T, STOP`

EXAMPLE

The current position (commanded) of axis 2 is unknown. It is known, however, that we want to move axis 2 8000 counts in the negative direction (that is, -8000 counts from the current position). The move should be accomplished with an acceleration of 1.0 counts/(200µs)^2 and a target slew rate of -3.5 counts/200µs.

```
REL_AXMOVE (0x2,1.0,-8000,3.5)
```

REL_AXMOVE_S

FUNCTION Relative S-Curve Axis Move with Trapezoidal Trajectory

EXECUTION 200 microseconds

SYNTAX `REL_AXMOVE_S (n, acc`$_1$`, pos`$_1$`, vel`$_1$`, ... , acc`$_8$`, pos`$_8$`, vel`$_8$`)`

USAGE DSPL (Motion), Host (command code: 75h)

ARGUMENTS

n — bit coding of the specified axis(es)

acc_x — unsigned value specifying the acceleration/deceleration for axis x

$0 \le acc_x \le 1.999969$ counts/(200µs)^2

pos_x — relative position for axis x

$-2147483648 \le pos_x \le 2147483647$ counts

vel_x — unsigned target velocity for axis x

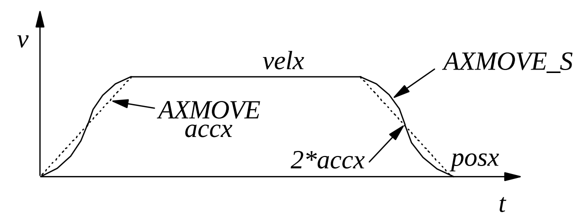
$0 \le vel_x \le 255.99998$ counts/200µs

When used in DSPL, arguments acc_x, pos_x, and vel_x may be selected as variables.

DESCRIPTION

The `REL_AXMOVE_S` RTC allows for s-curve command generation with relative (to current position) endpoint position, slew rate velocity and acceleration for each axis. This command is suitable for linear moves where s-curve acceleration is desired.

REL_AXMOVE_S cont.



The figure above illustrates the velocity profile of the `REL_AXMOVE_S` along with the linear velocity ramp of the `REL_AXMOVE` command. With `REL_AXMOVE_S`, the acceleration will reach a value of 2*accx for a maximum (see above figure).

SEE ALSO `AXMOVE, AXMOVE_S, AXMOVE_T, REL_AXMOVE, REL_AXMOVE_T, STOP`

EXAMPLE

The current position (commanded) of axis 2 is unknown. It is known, however, that we want to move axis 2 8000 counts in the negative direction (that is, -8000 counts from the current position). The move should be accomplished with an acceleration of 1.0 counts/(200µs)^2 and a target velocity of (unsigned) 3.5 counts/200µs.

```
REL_AXMOVE_S (0x2, 1.0, -8000, 3.5)
```

REL_AXMOVE_SLAVE

FUNCTION Superimposes a Relative Axis Move onto a Slave Engaged in Gearing

EXECUTION 200 microseconds

SYNTAX `REL_AXMOVE_SLAVE (n, acc, rel_pos, rel_vel)`

USAGE DSPL (Motion), Host (command code: AEh)

ARGUMENTS

n	bit coding the axes involved
acc	relative move acceleration
rel_pos	position value relative to current position
rel_vel	velocity value relative to current velocity

When used in DSPL, arguments acc, rel_pos and rel_vel may be selected as variables.

DESCRIPTION

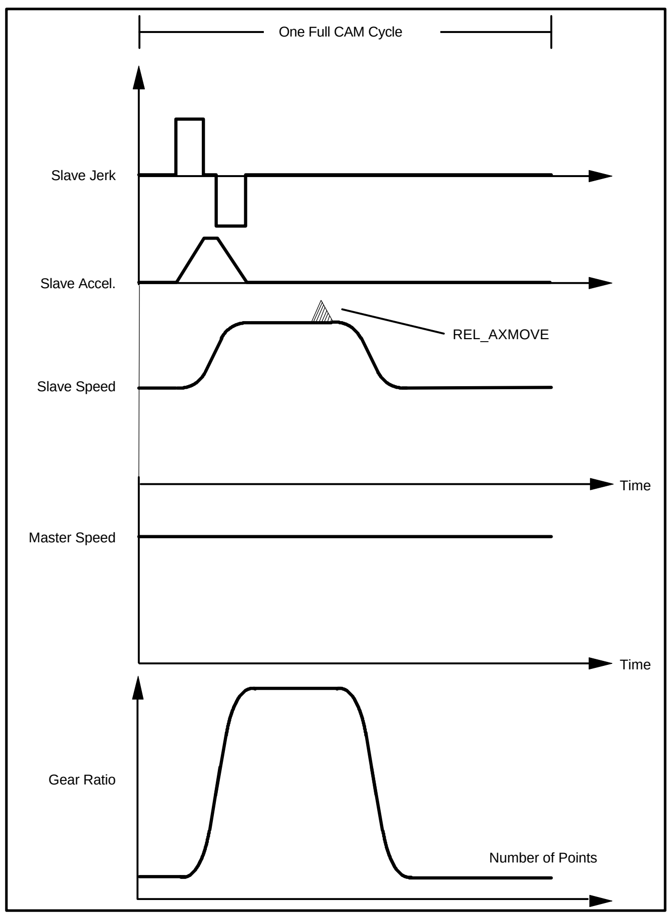
This command is similar to `AXMOVE` with two exceptions. First, it is relative not absolute; and second, it works only on the slave axis(es) involved in electronically geared or cam applications. This command allows the slave to momentarily disengage from the gearing process and compensate for its position short comings.

SEE ALSO `CAM, CAM_OFF, CAM_OFF_ACC, CAM_POS, CAM_PROBE, GEAR, GEAR_OFF, GEAR_OFF_ACC, GEAR_POS, GEAR_PROBE, SYNC`

APPLICATION

General master/slaving in particular flying shear applications can benefit from this instruction. Flying shear with registration marks is handled similarly to that of synchronous cutting. That is, the measured cutting error is used in the next cycle as an added function to compensate for the motion's shortcomings.

REL_AXMOVE_SLAVE cont.



REL_AXMOVE_T

FUNCTION Time-Based Relative Axis Move with Trapezoidal Trajectory

EXECUTION 200 microseconds

SYNTAX `REL_AXMOVE_T (n, acc`$_1$`, pos`$_1$`, tm`$_1$`, ... , acc`$_8$`, pos`$_8$`, tm`$_8$`)`

USAGE DSPL (Motion), Host (command code: 78h)

ARGUMENTS

n — bit coding of the specified axis(es)

acc_x — unsigned value specifying the acceleration/deceleration for axis x

$0 \le acc_x \le 1.999969$ counts/(200µs)^2

pos_x — relative position for axis x

$-2147483648 \le pos_x \le 2147483647$ counts

tm_x — motion time for axis x

$0 \le tm_x \le 5000000$ (200µs)

Note: The time argument, tm_x, is an unsigned value with a unit of 200usec.

When used in DSPL, arguments acc_x, pos_x, and tm_x may be selected as variables.

DESCRIPTION

The `REL_AXMOVE_T` RTC allows for trapezoidal command generation with relative (to current position) endpoint position, acceleration, and time to complete the move for each axis. This

REL_AXMOVE_T cont.

command is suitable for linear moves where relative endpoint position and motion time are the specifying parameters.

The `REL_AXMOVE_T` command is similar to `REL_AXMOVE`, with the exception that the velocity argument is replaced with a time argument. `REL_AXMOVE_T` will automatically calculate a suitable slew rate velocity to achieve the programmed relative endpoint position in the programmed amount of time, following a trapezoidal velocity profile (similar to `REL_AXMOVE`).

SEE ALSO `REL_AXMOVE, REL_AXMOVE_S, AXMOVE, AXMOVE_S, AXMOVE_T, STOP`

EXAMPLE

The current position (commanded) of axis 4 is unknown. It is known, however, that we want to move axis 4 10000 counts in the negative direction (that is, -10000 counts from the current position). The move should be accomplished with an acceleration of 1.0 counts/(200µs)^2 and be completed in 350msec (1750*200µsec).

```
REL_AXMOVE_T (0x8, 1.0, -10000, 1750)
```

RESET

FUNCTION Reset Mx4

EXECUTION 200 microseconds

SYNTAX `RESET (AAh, AAh)`

USAGE DSPL (Motion), Host (command code: 72h)

ARGUMENTS

AAh reset signature byte

DESCRIPTION

This command brings the servo controller card back to power-up state. Upon Mx4's reset completion, a host interrupt is generated via bit 4 of DPR locations [Mx4:7FEh] [Mx4 Octavia:1FFEh].

SEE ALSO none

APPLICATION

From time to time all systems may have to be software reset to allow for an initialization.

Command Sequence Example

No preparation is required before running this instruction.

EXAMPLE

Reset the Mx4 controller card.

```
RESET (0xAA, 0xAA)
```

RET

FUNCTION Return from Subroutine

EXECUTION 10 microseconds

SYNTAX `RET ( )`

USAGE DSPL (Motion)

ARGUMENTS

none

DESCRIPTION

This instruction is used to return from a called subroutine to the program which initiated the `CALL`. The program flow returns to the calling DSPL program after the `RET` instruction. The `RET` command is the last instruction of a subroutine.

SEE ALSO `CALL`

APPLICATION

See *Application Notes*

EXAMPLE

Return from a subroutine.

```
RET ( )
```

RUN_M_PROGRAM

FUNCTION Initiate DSPL Program Execution

EXECUTION 10 microseconds

SYNTAX

```
RUN_M_PROGRAM (program)
or
RUN_M_PROGRAM (program1, program2)
```

USAGE DSPL (PLC)

ARGUMENTS

program The program label of the Motion program to be run.

DESCRIPTION

Mx4 can run up to two Motion Programs on Mx4 and three on Mx4 Octavia simultaneously. If the user attempts to run more than two motion programs or Mx4's program buffer is full, an interrupt is generated to the host. Bit 0 of location 0Fh in the Dual Port RAM is set to 1 and bit 5 of the interrupt register 2 is also set.

SEE ALSO STOP_ALL_M_PROGRAM, STOP_M_PROGRAM

APPLICATION

See *DSPL Application Notes*

EXAMPLE

Begin execution of the DSPL programs "PROG_1" and "PROG_2".

```
RUN_M_PROGRAM (PROG_1)
RUN_M_PROGRAM (PROG_2)
```

SIGN

FUNCTION Find the Sign of a Constant or a Variable Value.

EXECUTION 10 microseconds

SYNTAX `SIGN(valu)` *or* `-SIGN(valu)`

USAGE DSPL (PLC, Motion)

ARGUMENTS

valu A constant
or a
variable (`VAR1` through `VAR128`)

DESCRIPTION

This function finds the sign of a constant or a variable value. The value returned is set as follows:

`SIGN`(valu) = -1	if valu < 0
`SIGN`(valu) = 0	if valu = 0
`SIGN`(valu) = +1	if valu > 0.

If a minus sign appears to the left of the SIGN function, the number returned by `SIGN` is multiplied by -1.

Note: This function can only be used with a variable assignment statement. For example:

```
VAR55 = SIGN(-88.43)
```

SEE ALSO `ABS, FRAC, INT, SQRT`

EXAMPLE

The first example finds the sign of the value stored in `VAR13` and stores the result in `VAR47`:

```
VAR47 = SIGN(VAR13)
```

The second example finds the sign of -71.482 and stores the result (-1) in `VAR31`:

```
VAR31 = SIGN(-71.482)
```

SIN

FUNCTION Calculate the Sine of a Constant or a Variable Value.

EXECUTION 75 microseconds

SYNTAX `SIN(valu)` *or* `-SIN(valu)`

USAGE DSPL (PLC, Motion)

ARGUMENTS

valu A constant real number
or a
variable (`VAR1` through `VAR128`)

DESCRIPTION

This mathematical function calculates the sine of a constant or a variable value specified in radians. If *valu* is a constant and a minus sign appears to the left of the `SIN` function, the result of the sine calculation is multiplied by -1.

Note: This function can only be used with a variable assignment statement. For example:

```
VAR34 = SIN(-1.72)
```

SEE ALSO `ARCTAN, COS, TAN`

EXAMPLE

The first example calculates the sine of the value stored in `VAR17` and stores the result in `VAR42`:

```
VAR42 = SIN(VAR17)
```

The second example finds the sine of 2.45 radians and stores the result (0.637764702) in `VAR37`:

```
VAR37 = SIN(2.45)
```

SINE_OFF

FUNCTION Turn Off Circular Interpolation Sine Table

EXECUTION 10 microseconds

SYNTAX `SINE_OFF (n)`

USAGE DSPL (PLC, Motion)

ARGUMENTS

n — bit coding of the axes for which sine tables will be disabled

DESCRIPTION

This instruction turns off (clears) the Mx4 position and velocity sine tables involved in circular interpolation. This way, the machine compensation table will be the only means of contouring.

SEE ALSO `CIRCLE`, `SINE_ON`, `TABLE_OFF`, `TABLE_ON`

APPLICATION

See *Application Notes*

EXAMPLE

Turn the sine table off for axes three and four.

```
SINE_OFF (0xC)
```

SINE_ON

FUNCTION Turn On Circular Interpolation Sine Table

EXECUTION 10 microseconds

SYNTAX `SINE_ON (n)`

USAGE DSPL (PLC, Motion)

ARGUMENTS

n bit coding of the axes for which sine table is enabled

DESCRIPTION

This instruction turns on (reactivates) the Mx4 position and velocity sine tables involved in circular interpolation. This instruction is executed after the execution of TURN OFF SINE TABLE.

SEE ALSO `CIRCLE`, `SINE_OFF`, `TABLE_OFF`, `TABLE_ON`

APPLICATION

See *Application Notes*

EXAMPLE

Enable the sine table for axes one, two, and three.

```
SINE_ON (0x7)
```

SQRT

FUNCTION Calculate the Positive Square Root of a Constant or Variable Value.

EXECUTION 75 microseconds

SYNTAX `SQRT(valu)` *or* `-SQRT(valu)`

USAGE DSPL (PLC, Motion)

ARGUMENTS

valu — A constant real number >= 0
or a
variable (`VAR1` through `VAR128`)

DESCRIPTION

This mathematical function calculates the square root of a constant or a variable value. If *valu* is a constant, it must be a constant >= 0 otherwise an error will be returned. If *valu* is a variable, the function will return the square root of the value stored in the variable if that value >= 0. Otherwise a value of zero is returned. If *valu* is a constant and a minus sign appears to the left of the `SQRT` function, the result of the square root calculation is multiplied by -1.

Note: This function can only be used with a variable assignment statement. For example:

```
VAR2 = SQRT(32.97)
```

SEE ALSO `ABS, FRAC, INT, SIGN`

EXAMPLE

The first example calculates the square root of the value stored in `VAR17` and stores the result in `VAR42`:

```
VAR42 = SQRT(VAR17)
```

The second example finds the square root of 12.75 and stores the negated result (-3.570714214) in `VAR16`:

```
VAR16 = -SQRT(12.75)
```

START

FUNCTION Start Contouring Motion

EXECUTION 10 microseconds

SYNTAX `START (n)`

USAGE DSPL (Motion), Host (command code: 6Dh)

ARGUMENTS

n bit coding of the specified axis(es)

DESCRIPTION

This command starts the motion (simultaneously) for the specified axes included in 2nd order and cubic spline contouring. `START` applies to contouring only.

Note: `START` will be ignored if contouring is in progress.

SEE ALSO `STOP, VECCHG`

APPLICATION

This command must be used in all 2nd order and ring buffer cubic spline contouring applications to start contouring with selected axes.

For 2nd Order Contouring Only

This command can be overwritten by `VECCHG` which redefines the axes involved in the contouring process. For example, `START` starts the contouring of axes 1, 3, and 4. If in the course of contouring, a `VECCHG` is received (with argument) specifying axes 1, 2, and 3, the new contouring points in the ring buffer will be used for the newly defined axes. Please also see `VECCHG`.

START cont.

Command Sequence Example

```
.                ;load ring buffer with positions and velocities
.
MAXACC ( )       ;make sure system can stop
CTRL ( )         ;set the gains
KILIMIT ( )
BTRATE ( )       ;set the block transfer rate
EN_BUFBRK ( )    ;set the breakpoint in the ring buffer
.
.
START ( )        ;start contouring
```

EXAMPLE

Start contouring motion in axes 2, 3, and 4.

```
START (0xE)
```

STEPPER_ON

Stp4 option command

FUNCTION Select Servo/Stepper Axes

EXECUTION 200 microseconds

SYNTAX `STEPPER_ON (n)`

USAGE DSPL (Motion), Host (command code: 8Dh)

ARGUMENTS

n — bit coding the axes selected as stepper axes (the remaining axes are servo axes)

DESCRIPTION

This command requires the Stp4 add-on card. `STEPPER_ON` allows the user to select the axes which are stepper control axes. The axes not selected by the n argument remain servo control axes.

EXAMPLE

Select axes 1 and 2 as stepper control axes.

```
STEPPER_ON (0x3)
```

STOP

FUNCTION Stop Motion

EXECUTION 75 microseconds

SYNTAX `STOP (n)`

USAGE DSPL (Motion), Host (command code: 6Eh)

ARGUMENTS

n bit coding of the specified axis(es)

DESCRIPTION

This command stops the motion of all specified axes simultaneously. To stop motion, the servo control card uses the programmed values for maximum acceleration / deceleration. Upon receipt of `STOP`, the servo controller aborts the current command. The host is responsible for clearing the ring buffer of any remaining commands if the axis(es) stopped was involved in contouring motion.

Note 1: An emergency stop signal, `ESTOP_ACC`, will perform a hardware stop. This is an open collector input signal which is active low and is shared between all of the controller cards.

Note 2: `STOP` will be ignored if the maximum acceleration / deceleration is equal to zero (e.g., `MAXACC` not issued).

If an axis is halting to a stop from a previously executed `STOP RTC` or active `ESTOP_ACC` input, Mx4 will ignore any motion commands (`AXMOVE`, `REL_AXMOVE`, `START` or `VELMODE`) and will report an "RTC Command Ignored" interrupt to the host. The above motion commands should not be sent to Mx4 for a halting axis until the axis motion has come to a stop.

SEE ALSO `MAXACC`, `START`

STOP cont.

APPLICATION

For all applications involving bringing speed to zero in the quickest possible manner.

Command Sequence Example

```
MAXACC ( )       ;set the maximum accel. so system can be stopped
CTRL ( )         ;set the gains
KILIMIT ( )
BTRATE ( )       ;set the block transfer rate
EN_BUFBRK ( )    ;set the breakpoint in the ring buffer
.
.
STOP ( )         ;stop the motion
.                ;upon completion of stop (command) trajectory
.                ;Mx4 generates motion complete interrupt
```

EXAMPLE

Bring the motion of axes 1 and 4 to a halt.

```
STOP (0x9)
```

STOP_ALL_M_PROGRAM

FUNCTION Terminate Execution of All DSPL Motion Programs

EXECUTION 200 microseconds

SYNTAX `STOP_ALL_M_PROGRAM ( )`

USAGE DSPL (PLC, Motion)

ARGUMENTS

none

DESCRIPTION

This instruction terminates the execution of all running DSPL Motion programs. DSPL Motion programs may be re-initiated via additional `RUN_M_PROGRAM` commands in the PLC program.

The `STOP_ALL_M_PROGRAM` command will also stop the motion (if any) of all axes with the programmed `MAXACC` acceleration.

SEE ALSO `MAXACC, RUN_M_PROGRAM, STOP_M_PROGRAM`

APPLICATION

See *Application Notes*

EXAMPLE

Stop the execution of all running Motion programs.

```
STOP_ALL_M_PROGRAM ( )
```

STOP_M_PROGRAM

FUNCTION Terminate Execution of DSPL Motion Program(s)

EXECUTION 50 microseconds

SYNTAX
```
STOP_M_PROGRAM (program)
or
STOP_M_PROGRAM (program1, program2)
```

USAGE DSPL (PLC, Motion)

ARGUMENTS

program The program label of the Motion program to be stopped

DESCRIPTION

The `STOP_M_PROGRAM` command is used to stop the execution of selected DSPL Motion programs. DSPL Motion programs may be re-initiated via additional `RUN_M_PROGRAM` commands in the PLC program.

SEE ALSO `RUN_M_PROGRAM, STOP_ALL_M_PROGRAM`

APPLICATION

See *Application Notes*

EXAMPLE

Stop the execution of DSPL programs TEST1 and TEST2.

```
STOP_M_PROGRAM (TEST1, TEST2)
```

SYNC

FUNCTION Master / Slave Select

EXECUTION 10 microseconds

SYNTAX `SYNC (m)`

USAGE DSPL (motion), Host (command code: 87h)

ARGUMENTS

m selects the Master / Slave status of the Mx4 card

m = 0 : Mx4 is configured as a Master
m <> 0 : Mx4 is configured as a Slave

DESCRIPTION

If more than one Mx4 card is to be used in a system and card-to-card synchronization is required, the `SYNC` command should be used. `SYNC` allows multiple Mx4 cards to operate in synchronization within a system by specifying a single Master and the remaining card(s) as Slaves. If only one Mx4 is used in a host computer system, that Mx4 must be configured as a Master.

Note: Mx4 powers-up and resets to a default Master status.

In addition to configuring the Mx4 cards with `SYNC` (for multiple card systems), a cable jumper must be included on the J5 connector of each of the boards. The cable must be wired such that the MASTER signal from the Master Mx4 connects to the SLAVE signal of each of the Slave Mx4(s) (see *Mx4 User's Guide, Installing Your Mx4*).

SEE ALSO none

SYNC cont.

APPLICATION

This command is used in applications where tight coordination of more than four axes (when using Mx4s) or eight axes (when using Mx4 Octavias) is required. This command essentially slaves several Mx4 cards to a single Master Mx4. Applications involving many axes contouring may benefit from this command.

Command Sequence Example

This command must be executed immediately after the initialization. Please remember that the default value for m is zero (i.e., the card is initialized as a Master).

EXAMPLE

Configure the Mx4 controller as a slave in a multi-Mx4 synchronized system.

```
SYNC (0x1)
```

TABLE_OFF

FUNCTION Disable Position and Velocity Circular Interpolation Compensation Tables

EXECUTION 10 microseconds

SYNTAX `TABLE_OFF (n)`

USAGE DSPL (PLC, Motion)

ARGUMENTS

n — bit coding of the axes for which the position and velocity compensation tables are disabled

DESCRIPTION

The `TABLE_OFF` command turns 'off' or disables the position and velocity compensation tables for the specified axes. After a `TABLE_OFF` command for an axis, any circular interpolation involving that axis will use the sine tables only for the circular interpolation.

Note: Before executing a `TABLE_OFF` command, it is important that the sine table for the axis is enabled.

SEE ALSO `CIRCLE`, `SINE_ON`, `SINE_OFF`, `TABLE_ON`

APPLICATION

See *Application Notes*

EXAMPLE

Disable the compensation tables for axes two, three, and four:

```
TABLE_OFF (0xE)
```

TABLE_ON

FUNCTION Enable Position and Velocity Circular Interpolation Compensation Tables

EXECUTION 10 microseconds

SYNTAX `TABLE_ON (n)`

USAGE DSPL (PLC, Motion)

ARGUMENTS

n bit coding of the axes for which the position and velocity compensation tables are enabled

DESCRIPTION

This instruction will compensate for velocity and position inaccuracies or nonlinearities of the system's mechanical parts involved in circular interpolation. The compensation tables must be downloaded to Mx4 before execution of the `TABLE_ON` command.

Note: Position and velocity compensation tables are 1024 locations long. There is a corresponding position and velocity compensation table for each axis. For both position and velocity tables, each point is a 15-bit two's complement value, hence it represents an absolute 14-bit value. Mx4 initializes the tables' values to zeros. If the table is loaded with m points, where m is less than 1024, the remaining points will be zero. If the table is loaded with more than 1024 values, the additional points will be ignored.

SEE ALSO `CIRCLE`, `SINE_ON`, `SINE_OFF`, `TABLE_OFF`

APPLICATION

See *Application Notes*

EXAMPLE

Enable the compensation tables for axes two and four:

```
TABLE_ON (0xA)
```

TABLE_P, TABLE_V

IDENTIFIER

IDENTIFIER DSPL Table

SYNTAX

```
TABLE_P(index) = valu1
TABLE_V(index) = valu2
var = TABLE_P(index)
var = TABLE_V(index)
```

USAGE DSPL (PLC, Motion)

ARGUMENTS

index	0 < constant integer value < 4095 or a DSPL variable value (VAR1 through VAR128)
valu1	$-2147483648 \le \text{valu1} \le 2147483647$ or a DSPL variable value (VAR1 through VAR128)
valu2	$0 \le \text{valu2} \le 255.99998$ or a DSPL variable value (VAR1 through VAR128)
var	DSPL variable value (VAR1 through VAR128)

DESCRIPTION

The DSPL tables TABLE_P and TABLE_v can be used to store integer and fractional values in a DSPL program. Values in TABLE_P are stored in the position format (32-bit two's complement integer values), while values in TABLE_V are stored in the velocity format (25-bit two's complement values sign extended to 32-bits with the least significant 16 bits representing the fractional value.)

Note The DSPL tables, cam, internal cubic spline, and position/velocity compensation tables share overlapping data space in Mx4.

TABLE_P, TABLE_V cont. **IDENTIFIER**

Note The fractional portion of any values stored in `TABLE_P` will be truncated. Values stored in `TABLE_V` can have a maximum absolute value of 256.

EXAMPLES

The first example stores the value 12 (truncated from 12.3) into the table at index 13:

```
TABLE_P(13) = 12.3
```

The second example stores the value in `VAR12` in the table at the location indexed by the value in `VAR1`:

```
TABLE_V(VAR1) = VAR12
```

The third example retrieves the value in the table at the location indexed by the value in `VAR17` and stores the value in `VAR28`:

```
VAR28 = TABLE_V(VAR17)
```

TABLE_SEL

FUNCTION Select Compensation Table

EXECUTION 50 microseconds

SYNTAX `TABLE_SEL (n, tb1, ... , tb8)`

USAGE DSPL (Motion), Host (command code: A2h)

ARGUMENTS

n bit coding the axes involved

tb_x specifies the compensation table to be used for axis x

$1 \le tb_x \le 8$

DESCRIPTION

The `TABLE_SEL` command allows the user to arbitrarily select the compensation table for the axis(es) in question. More than one axis may use a compensation table.

SEE ALSO `CIRCLE`, `TABLE_OFF`, `TABLE_ON`

EXAMPLE

Axes 1 and 2 are to use compensation table 2, while axes 3 and 4 use compensation table 1.

```
TABLE_SEL (0xF,2,2,1,1)
```

TAN

FUNCTION Calculate the Tangent of a Constant or a Variable Value.

EXECUTION 100 microseconds

SYNTAX `TAN(valu)`

USAGE DSPL (PLC, Motion)

ARGUMENTS

valu — A constant real number or a variable (`VAR1` through `VAR128`)

DESCRIPTION

This mathematical function calculates the tangent of a constant or a variable value specified in radians. If *valu* is a constant and a minus sign appears to the left of the `TAN` function, the result of the tangent calculation is multiplied by -1.

Note: This function can only be used with a variable assignment statement. For example:

```
VAR11 = TAN(1.163)
```

SEE ALSO `ARCTAN, COS, SIN`

EXAMPLE

The first example calculates the tangent of the value stored in `VAR51` and stores the result in `VAR14`:

```
VAR14 = TAN(VAR51)
```

The second example finds the tangent of -2.009 radians and stores the result (2.134071211) in `VAR24`:

```
VAR24 = TAN(-2.009)
```

TIMER, TIMER1, …, TIMER4 **IDENTIFIER**

FUNCTION General Purpose DSPL Timer State Variable

SYNTAX [Mx4 / Mx42]

`TIMER`

[Mx4 Octavia]

`TIMER1, TIMER2, TIMER3, TIMER4`

USAGE DSPL (PLC, Motion)

DESCRIPTION

`TIMER` is a DSPL identifier, which in conjunction with `TIMER_RESET` provides a general purpose DSPL timer with time increments of 200 µsec. `TIMER` may be reset to 0 via the `TIMER_RESET` command.

SEE ALSO TIMER_RESET

EXAMPLE

Implement a one second delay with `TIMER` on an Mx4.

```
TIMER_RESET (  )
WAIT_UNTIL (TIMER >= 5000)
```

TIMER_RESET

FUNCTION Reset General Purpose DSPL Timer

EXECUTION 10 microseconds

SYNTAX [Mx4 / Mx42]

```
TIMER_RESET ( )
```

[Mx4 Octavia]

```
TIMER_RESET (n)
```

USAGE DSPL (PLC, Motion)

ARGUMENTS

n bit coding of specified timer(s) (only applies to Octavia)

DESCRIPTION

This command resets the single DSPL timer on an Mx4, or any of the four timers on an Mx4 Octavia.

SEE ALSO TIMER, TIMER1, …, TIMER4

EXAMPLE

Implement a one second delay with `TIMER` on an Mx4.

```
TIMER_RESET (  )
WAIT_UNTIL (TIMER >= 5000)
```

TRQ_LIMIT

FUNCTION DAC Output Voltage Limit

EXECUTION 200 microseconds

SYNTAX `TRQ_LIMIT (n, val`$_1$`, ... , val`$_8$`)`

/USAGE DSPL (Motion), Host (command code: 5Bh)

ARGUMENTS

n	bit coding of the specified axis(es)
val_x	DAC output voltage (abs) limit for axis x
	$-10.0 <= val_x <= 9.9997$ volts

When used in DSPL, the argument val_x may be selected as a variable.

DESCRIPTION

The `TRQ_LIMIT` command specifies a torque limit (by means of output voltage limiting) value ranging from 0 volts (no output) to +/-10 volts (full swing) with a resolution of approximately 0.3 millivolts.

The Mx4 controller powers-up and resets to a default torque limit value allowing full output voltage swing.

SEE ALSO none

APPLICATION

This command can be used in applications where an axis torque needs to be limited, such as packaging or material handling.

Command Sequence Example

No preparation is required before running this instruction.

EXAMPLE

Limit the output voltage swing for axis 2 to +/- 7.5 volts.

```
TRQ_LIMIT (0x2, 7.5)
```

VAR1, ..., VAR128 — IDENTIFIER

IDENTIFIER DSPL Variable

USAGE DSPL (PLC, Motion)

DESCRIPTION

The 128 DSPL variables hold floating point real numbers that can be stored, retrieved, and manipulated by the DSPL programmer.

EXAMPLES

The DSPL variables can be used to do the following:

- specify the value of an argument in a DSPL command or function:

```
AXMOVE(0x1, VAR1, VAR23, VAR14)
VAR1 = SQRT(VAR32)
```

- store constant numbers:

```
VAR3 = -9385.38
VAR5 = 0x34
```

- assign the value of one variable to another:

```
VAR13= VAR29
```

- perform intermediate computations:

```
VAR23 = VAR2 / 23.78
VAR51 = VAR32 * VAR12
```

- retrieve/store a value from/to a DSPL tables (`TABLE_P` and `TABLE_V`):

```
VAR23 = TABLE_V(332)
TABLE_P(123) = VAR2
```

- provide an index into one of the DSPL tables:

```
TABLE_V(VAR7) = 3.75
```

- provide bit register functionality

```
VAR4 = VAR55 & 0x1133
```

- specify one or both of the values in a conditional expression:

```
WAIT_UNTIL(VAR12 > VAR50)
```

VECCHG

FUNCTION 2nd Order Contouring Vector Change

EXECUTION 10 microseconds

SYNTAX `VECCHG (n, m)`

USAGE DSPL (Motion), Host (command code: 6Fh)

ARGUMENTS

n	bit coding of the specified axis(es) involved
m	value which represents the buffer position (in 8 byte offsets from the start of the buffer) where the number of axes involved in contouring must be changed to include only those axes coded by n

DESCRIPTION

Upon the execution of this command, the 2nd order contouring task assumes a new set of axes at the programmed pointer location.

Note: Three buffer levels are used to implement this instruction.

SEE ALSO `START`

APPLICATION

See `START`.

VECCHG cont.

Command Sequence Example

```
MAXACC ( )      ;set the maximum accel. so system can be stopped
CTRL ( )        ;set the gains
KILIMIT ( )
BTRATE ( )      ;set the block transfer rate
EN_BUFBRK ( )   ;set the buffer breakpoint interrupt
.
.
START ( )       ;start contouring for a selected number of axes
.               ;based on buffer breakpoint interrupt transfer more
.               ;points
VECCHG ( )      ;use points in ring buffer for a new set of axes
```

EXAMPLE

Begin 2nd order contouring in axes 1, 2, and 3 after the 23rd segment move command of the ring buffer.

```
VECCHG (0x7,23)
```

VECT4_PAR1, ..., VECT4_PAR8 IDENTIFIER

IDENTIFIER Vx4++ Parameter

USAGE DSPL (PLC, Motion)

DESCRIPTION

With the Vx4++ option, Vx4++ state variables are available in Mx4s' DSPL programming language. The source of the state variable is selected with the VIEWVEC command.

Name	**Description**
`VECT4_PAR1`	Vx4++ parameter 1
`VECT4_PAR2`	Vx4++ parameter 2
`VECT4_PAR3`	Vx4++ parameter 3
.	
.	
`VECT4_PAR8`	Vx4++ parameter 8

SEE ALSO `VIEWVEC`

EXAMPLE

The Vx4++ parameters can be used as follows:

- as one of the values used in conjunction with a DSPL arithmetic operation:

```
VAR12 = VECT4_PAR3 + 3000
```

- as one of the arguments in a DSPL conditional expression:

```
WHILE(VECT4_PAR1 > 100000)
```

VX4_BLOCK

VX4++ option command

FUNCTION Blocks Vx4++ commands

EXECUTION 200 microseconds

SYNTAX `VX4_BLOCK (m, blk1, blk2)`

USAGE DSPL (Motion), Host (command code: 84h)

ARGUMENTS

m — bit coding of the specified axis groups

m = 0x3	axes one, two
m = 0xC	axes three, four
m = 0xF	axes one, two, three, four

blk_1 — block code for axes one, two
blk_2 — block code for axes three, four

blk_x = 0	Vx4++ block disabled
blk_x = 1	Vx4++ block enabled

DESCRIPTION

This command is used to block some of the VX4++ commands so that those commands may not be accidentally executed. The user is responsible for disabling the block command in order to execute one of the commands listed below (SEE ALSO).

SEE ALSO `CURR_LIMIT, CURR_OFFSET, ENCOD_MAG, MOTOR_TECH, PWM_FREQ`

APPLICATION

See *Vx4++ User's Guide*

EXAMPLE

Enable the Vx4++ command blocking for all four axes.

```
VX4_BLOCK (0xF, 1, 1)
```

VEL1, …, VEL8 — IDENTIFIER

IDENTIFIER Actual Velocity State Variable

USAGE DSPL (PLC, Motion)

DESCRIPTION

A actual velocity state variable holds a value that represents the current velocity (in encoder edge counts/200µs) of the specified axis:

Name	Description
`VEL1`	axis 1 actual velocity
`VEL2`	axis 2 actual velocity
`VELx`	axis x actual velocity
.	
.	
`VEL8`	axis 8 actual velocity

SEE ALSO `CVEL1`

EXAMPLE

The actual velocity state variables can be used as follows:

- as one of the values used in conjunction with a DSPL arithmetic operation:

  ```
  VAR12 = VEL2 - 1.5
  ```

- as one of the arguments in a DSPL conditional expression:

  ```
  WHILE(VEL4 > 1.5)
  ```

VELMODE

FUNCTION Velocity Mode

EXECUTION 100 microseconds

SYNTAX `VELMODE (n, vel1, ... , vel8)`

USAGE DSPL (Motion), Host (command code: 70h)

ARGUMENTS

n	bit coding of the specified axis(es)
vel_x	target velocity for axis x
	$-256 \le vel_x \le 255.99998$ counts/200µs

When used in DSPL, argument vel_x may be selected as a variable.

DESCRIPTION

Upon the execution of this command, a velocity loop for the specified axes will be closed. The velocity loop uses the same gains as those specified using the control law command. `VELMODE` uses the `MAXACC` maximum acceleration / deceleration value to accelerate or decelerate to the desired velocity.

Note : `VELMODE` will be ignored if the maximum acceleration / deceleration is equal to zero (e.g., `MAXACC` not issued).

SEE ALSO MAXACC

APPLICATION

This instruction is useful in all general purpose velocity control applications. Please remember that although `VELMODE` primarily regulates speed, the outer loop is still position. This means that while regulating speed, Mx4 continually tries to zero the position error.

Command Sequence Example

```
MAXACC ( )       ;set the maximum accel. so system can be stopped
CTRL ( )         ;set the gains
KILIMIT ( )
.
VELMODE ( )
```

EXAMPLE

Engage axis 2 in velocity mode with a velocity of 3.71 counts/200 µs.

```
VELMODE (0x2,3.71)
```

VIEWVEC

Vx4++ option command

FUNCTION Specify Vx4++ State Variables to View

EXECUTION 200 microseconds

SYNTAX `VIEWVEC (n, m)`

USAGE DSPL (Motion), Host (command code: 83h)

ARGUMENTS

n	bit coding of the specified axis(es)
m	value specifying state variable

m=0	Iqs error
m=1	Ids error
m=2	Iqs feedback
m=3	Ids feedback
m=4	Iqs command
m=5	Ir feedback
m=6	Is feedback
m=7	It feedback

DESCRIPTION

This command selects the Vx4++ state variable which is available in the Mx4 Dual Port RAM and also with the `VECT4_PARx` DSPL identifiers.

As is evident above, only 1 variable may be "viewed" per axis at any given time.

SEE ALSO none

APPLICATION

See *Vx4++ User's Guide*

EXAMPLE

Change the Vx4++ state variable selection to Ids feedback for axis 1. Any subsequent VECT4_PAR1 accesses will yield the axis 1 Ids feedback value.

```
VIEWVEC (0x1, 3)
```

WAIT_UNTIL

FUNCTION Halt Program Execution Until Condition is True.

EXECUTION Depends on user arguments

SYNTAX `WAIT_UNTIL (conditional expression)`

USAGE DSPL (PLC, Motion)

ARGUMENTS

conditional expression

The conditional expression must be boolean, equating to True or False. The conditional expression may consist of multiple boolean conditions ANDed or ORed together. The conditional expression operators are:

>	greater than
<	less than
>=	greater than or equal
< =	less than or equal
= =	equal
!=	not equal
AND	logical AND
OR	logical OR
&	bit-wise AND

The conditional expression is enclosed via sets of parentheses. Nested parentheses may be used when multiple boolean conditions are used or more complex conditional expressions are implemented.

Note: If nested parentheses are not used to indicate evaluation precedence in a conditional expression, the expression will be evaluated from left-to-right.

WAIT_UNTIL cont.

DESCRIPTION

This instruction controls the flow of the program. If `WAIT_UNTIL` statements are used, Mx4 will wait until the boolean condition is True, then it executes the instructions after the `WAIT_UNTIL` statement.

SEE ALSO none

APPLICATION

All general motion application programs.

EXAMPLE

Halt program execution or 'wait' for the axis four command velocity to be greater than -4.55 and an active IN0 (1) input before continuing.

```
WAIT_UNTIL ((CVEL4 > -4.55) AND (INP1_REG & 0x0001))
```

WAIT_UNTIL_RTC

FUNCTION Halt Program Execution Until RTC Signal Is Received

EXECUTION Runs until an RTC is detected

SYNTAX `WAIT_UNTIL_RTC ( )`

USAGE DSPL (Motion)

ARGUMENTS

none

DESCRIPTION

After execution of the `WAIT_UNTIL_RTC` command, the DSPL Motion program waits until Mx4 receives (from the host) the RTC command SIGNAL_DSPL.

SEE ALSO none

APPLICATION

All generic motion application programs.

EXAMPLE

Halt program execution until the SIGNAL_DSPL RTC is received from the host.

```
WAIT_UNTIL_RTC ( )
```

WEND

FUNCTION Designates End of WHILE-WEND Structure

EXECUTION 10 microseconds

SYNTAX

```
WHILE (conditional expression)
    program code to execute while WHILE condition is True
WEND
```

USAGE DSPL (PLC, Motion)

ARGUMENTS

none

DESCRIPTION

The `WHILE-WEND` structure is used for conditional repeat loop program execution. `WEND` designates the last line of the `WHILE-WEND` structure. A `WEND` statement must be included with every `WHILE` statement.

SEE ALSO `WHILE`

APPLICATION

See *Application Notes*

EXAMPLE

While the following error of axis two is less than 50 counts, monitor the velocity of axis one. If the command velocity of axis one is greater than 2.0, bring axis one to a halt.

```
WHILE (ERR2 < 50)
        IF (CVEL1 > 2.0)
                STOP (0x1)
        ENDIF
WEND
```

WHILE

FUNCTION Designates Beginning of WHILE - WEND Structure

EXECUTION 200 microseconds

SYNTAX

```
WHILE (conditional expression)
    program code to execute while condition is True
WEND
```

USAGE DSPL (PLC, Motion)

ARGUMENTS

conditional expression

The conditional expression must be boolean, equating to True or False. The conditional expression may consist of multiple boolean conditions ANDed or ORed together. The conditional expression operators are:

>	greater than
<	less than
>=	greater than or equal
<=	less than or equal
= =	equal
!=	not equal
AND	logical AND
OR	logical OR
&	bit-wise AND

See 'DSPL Variables' for the complete list of variables which may be used in conditional expressions.

The conditional expression is enclosed via sets of parentheses. Nested parentheses may be used when multiple boolean conditions are used or more complex conditional expressions are implemented.

Note: If nested parentheses are not used to indicate evaluation precedence in a conditional expression, the expression will be evaluated from left-to-right.

WHILE cont.

For example,

```
WHILE ( (VAR1 > 100) AND (POS2 > 100) AND
(ERR1 == 200) OR (IN_REG1 & 0x3) AND
(CVEL1 > 10 ) )
```

This line is interpreted in DSPL as:

```
WHILE ( { { { [ (VAR1 > 100) AND (POS2 > 100) ]
AND (ERR1 == 200) } OR
(IN_REG1 & 0x3) }AND ( (CVEL1 > 10) })
```

DESCRIPTION

The `WHILE-WEND` structure allows for a repeating program loop based on a conditional expression. The program commands between the `WHILE` and `WEND` lines are executed while the conditional expression is TRUE. If the conditional expression evaluates FALSE, program execution jumps to the first command following the `WEND` command.

`WHILE-WEND` structures may be nested.

SEE ALSO

`WEND`

APPLICATION

See *Application Notes*

EXAMPLE

While the following error of axis two is less than 50 counts, monitor the velocity of axis one. If the command velocity of axis one is greater than 2.0, bring axis one to a halt.

```
WHILE (ERR2 < 50)
        IF (CVEL1 > 2.0)
                STOP (0x1)
        ENDIF
WEND
```

= OPERATOR

OPERATOR = (Assignment)

SYNTAX
```
var = valu1
or
tablename = valu2
```

USAGE DSPL (PLC, Motion)

ARGUMENTS

var	DSPL variable (`VAR1` through `VAR128`)
valu1	A constant real number, variable (`VAR1` through `VAR64`), state variable, ADC value, table value, function's return value, or the result of an operation
tablename	`TABLE_P` or `TABLE_V` (including index)
valu2	A constant real number or variable (`VAR1` through `VAR64`)

DESCRIPTION

This operator (=) is used to set the value of a DSPL variable. The assignment operator can also be used to assign either a constant or variable value to a location in `TABLE_P` or TABLE_V.

Note: This operation must be used when invoking any of DSPL's basic arithmetic operators, elementary math functions, or trigonometric functions.

SEE ALSO `+, -, *, /, ABS, ARCTAN, COS, FRAC, INT, SIGN, SIN, SQRT, TAN`

= cont. OPERATOR

EXAMPLE

The first example stores a constant in VAR32:

```
VAR32 = -9001.42
```

The second example stores the value of the command velocity of axis4 into VAR9:

```
VAR9 = CVEL1
```

The third example stores the result of the given addition in VAR11:

```
VAR11 = VAR21 + 22.3
```

The fourth example assigns the value stored at index 2019 of TABLE_V to VAR25:

```
VAR25 = TABLE_V(2019)
```

The fifth example stores a constant in TABLE_P at the index value specified by the value stored in VAR4:

```
TABLE_P(VAR4) = 7743
```

+ OPERATOR

OPERATOR + (Addition)

EXECUTION 100 microseconds

SYNTAX `valu1 + valu2`

USAGE DSPL (PLC, Motion)

ARGUMENTS

valu1 A constant real number,
variable (`VAR1` through `VAR128`),
state variable, or ADC value

valu2 A constant real number,
variable (`VAR1` through `VAR128`),
state variable, or ADC value

DESCRIPTION

The addition operator (+) is used to add two values. If a value is a variable, the value stored in the variable can be negated before performing the addition by inserting a minus sign (-) immediately before the variable name.

Note: This operator can only be used with a variable assignment statement. For example:

```
VAR15 = VAR2 + 12.5
```

Note: No more than one of the basic arithmetic operators (+, -, *, /) can appear on a single line of DSPL code: The following are therefore **NOT** valid lines of DSPL code:

```
VAR1 = VAR9 + VAR53 + 2.54
VAR2 = VAR9 + VAR3 * VAR4
```

+ cont. OPERATOR

Note: No more than one of the values to be added can be a DSPL state variable or ADC value. The following is therefore **NOT** a valid line of DSPL code:

```
VAR15 = ERR3 + POS1
```

SEE ALSO -, *, /

EXAMPLE

The first example adds two numbers, -9001.42 and 633.7 and stores the result in VAR31:

```
VAR31 = -9001.42 + 633.7
```

The second example adds 57 to the value stored in VAR22. The result is stored in VAR51:

```
VAR51 = 57 + VAR22
```

The third example negates the value stored in VAR13, negates the value in VAR29, and adds the two values. The result is stored in VAR29:

```
VAR29 = -VAR13 + -VAR29
```

The fourth example adds the command position of axis 3 to the value stored in VAR41. The result is stored in VAR14:

```
VAR14 = CPOS3 + VAR41
```

-   OPERATOR

OPERATOR **-** (Subtraction)

EXECUTION 100 microseconds

SYNTAX `valu1 - valu2`

USAGE DSPL (PLC, Motion)

ARGUMENTS

valu1 A constant real number,
variable (`VAR1` through `VAR128`),
state variable, or ADC value

valu2 A constant real number,
variable (`VAR1` through `VAR128`),
state variable, or ADC value

DESCRIPTION

The subtraction operator (**-**) is used to subtract one value from another. If a value is a variable, the value stored in the variable can be negated before performing the subtraction by inserting a minus sign (**-**) immediately before the variable name.

Note: This operator can only be used with a variable assignment statement. For example:

```
VAR25 = VAR52 - 99.2
```

Note: No more than one of the basic arithmetic operators (+, **-**, *, /) can appear on a single line of DSPL code: The following are therefore **NOT** valid lines of DSPL code:

```
VAR31 = VAR9 - VAR3 - 2.54
VAR27 = VAR9 + 132.3 - VAR4
```

- cont. OPERATOR

Note: No more than one of the values to be operated on can be a DSPL state variable or ADC value. The following is therefore **NOT** a valid line of DSPL code:

```
VAR4 = ERR2 - CVEL4
```

SEE ALSO +, *, /

EXAMPLE

The first example subtracts 1 from 0.041 and stores the result in `VAR60`

```
VAR60 = 0.041 - 1
```

The second example subtracts the value stored in `VAR2` from 44.4. The result is stored in `VAR2`:

```
VAR2 = 44.4 - VAR2
```

The third example negates the value stored in `VAR3`, then subtracts the value in `VAR12`. The result is stored in `VAR9`:

```
VAR9 = -VAR3 - VAR12
```

The fourth example subtracts the command velocity of axis 1 from the value stored in `VAR4`. The result is stored in `VAR49`:

```
VAR49 = VAR4 - CVEL1
```

* OPERATOR

OPERATOR * (Multiplication)

EXECUTION 100 microseconds

SYNTAX `valu1 * valu2`

USAGE DSPL (PLC, Motion)

ARGUMENTS

valu1 A constant real number,
variable (VAR1 through VAR128),
state variable, or ADC value

valu2 A constant real number,
variable (VAR1 through VAR128),
state variable, or ADC value

DESCRIPTION

The multiplication operator (*) is used to multiply one value by another. If a value is a variable, the value stored in the variable can be negated before performing the multiplication by inserting a minus sign (-) immediately before the variable name.

Note: This operator can only be used with a variable assignment statement. For example:

```
VAR4 = VAR25 * -8902
```

Note: No more than one of the basic arithmetic operators (+, -, *, /) can appear on a single line of DSPL code: The following are therefore **NOT** valid lines of DSPL code:

```
VAR12 = VAR59 * 22.86 * VAR5
VAR17 = 9 - VAR3 * VAR24
```

* cont. OPERATOR

Note: No more than one of the values to be operated on can be a DSPL state variable or ADC value. The following is therefore **NOT** a valid line of DSPL code:

```
VAR11 = CPOS1 - ERR4
```

SEE ALSO `+, -, /`

EXAMPLE

The first example multiplies two numbers 0.1751 and 0.441and stores the result in VAR64

```
VAR64 = 0.1751 * 0.441
```

The second example multiplies the value stored in VAR22 by -100. The result is stored in VAR2:

```
VAR2 = -100 * VAR22
```

The third example negates the value stored in VAR5, negates the value in VAR48, then multiplies the two resulting values. The result is stored in VAR39:

```
VAR39 = -VAR5 * -VAR48
```

The fourth example multiplies the actual velocity of axis 4 by the value stored in VAR7. The result is stored in VAR49:

```
VAR49 = VEL4 * VAR7
```

/ OPERATOR

OPERATOR **/** (Division)

EXECUTION 100 microseconds

SYNTAX `valu1 / valu2`

USAGE DSPL (PLC, Motion)

ARGUMENTS

valu1	A constant real number, variable (`VAR1` through `VAR128`), state variable, or ADC value
valu2	A constant real number or variable (`VAR1` through `VAR128`)

DESCRIPTION

The division operator (/) is used to divide one value by another. If a value is a variable, the value stored in the variable can be negated before performing the division by inserting a minus sign (-) immediately before the variable name.

Note: This operator can only be used with a variable assignment statement. For example:

```
VAR4 = VAR25 / -8902
```

Note: No more than one of the basic arithmetic operators (+, **-**, *, /) can appear on a single line of DSPL code: The following are therefore **NOT** valid lines of DSPL code:

```
VAR62 = VAR20 / 29 / 14.1
VAR1 = 9 + VAR10 / VAR2
```

/ cont. OPERATOR

Note: Only the numerator value (valu1) can be a DSPL state variable or ADC value. The following is therefore **NOT** a valid line of DSPL code:

```
VAR19 = VAR31 / CVEL4
```

SEE ALSO +, -, *

EXAMPLE

The first example divides **-**1.51 by 1111 and stores the result in `VAR60`

```
VAR60 = -1.51 / 1111
```

The second example divides the value stored in `ADC3` by 22.91. The result is stored in `VAR62`:

```
VAR62 = ADC3 / 22.91
```

The third example negates the value stored in `VAR55`, then divides the resulting value by the value stored in `VAR12`. The result is stored in `VAR3`:

```
VAR3 = -VAR55 / VAR12
```

The fourth example divides the actual position of axis 2 by the value stored in `VAR1`. The result is stored in `VAR9`:

```
VAR9 = POS2 / VAR1
```

~ OPERATOR

OPERATOR ~ (Bitwise Complement)

EXECUTION 60 microseconds

SYNTAX `~i_reg`

USAGE DSPL (PLC, Motion)

ARGUMENTS

i_reg — One of the DSPL interrupt registers (i.e. `ESTOP_REG, FERR_REG, FERRH_REG, INDEX_REG, MOTCP_REG, OFFSET_REG, POSBRK_REG, or PROBE_REG`)
or
One of the DSPL input registers (i.e. `INP1_REG or INP2_REG`)

DESCRIPTION

The bitwise complement operator (~) is used to find the complement of the contents of one of the DSPL interrupt or input registers before it is used in a DSPL conditional expression.

Note: This operator can only be used in a DSPL conditional expression inside of a DSPL conditional structure (i.e. `IF, WHILE, or WAIT_UNTIL`). For example:

```
WAIT_UNTIL(~FERR_REG & 0x02)
```

Note: The bitwise complement operator can only be used with the DSPL registers, and will **NOT** work with DSPL variables, state variables, or table values.

SEE ALSO `&, AND, OR, IF, WAIT_UNTIL, WHILE`

~ cont. OPERATOR

EXAMPLE

The conditional expression used in the `WAIT_UNTIL` statement below masks out all bits except bits 0 and 3 of the complemented index pulse interrupt register:

```
WAIT_UNTIL(~INDEX_REG & 0x09)
```

& OPERATOR

OPERATOR &(Bitwise AND)

EXECUTION 75 microseconds

SYNTAX `i_reg & mask_val`

USAGE DSPL (PLC, Motion)

ARGUMENTS

i_reg — One of the DSPL interrupt registers (i.e. `ESTOP_REG, FERR_REG, FERRH_REG, INDEX_REG, MOTCP_REG, OFFSET_REG, POSBRK_REG, or PROBE_REG`)
or
One of the DSPL input registers (i.e. `INP1_REG or INP2_REG`)

mask_val — A user defined bit mask that must be used in conjunction with the bitwise operator &. The mask follows the format 0x????, where ???? is a 16-bit hexadecimal value. For example, a mask value of 0x0204 will mask out all bits except bits 2 and 9.

DESCRIPTION

The bitwise AND operator (&) is used to mask selected bits in a DSPL interrupt or input register before it is used in a DSPL conditional expression.

Note: This operator is only used in a DSPL conditional expression inside of a DSPL conditional structure (i.e. `IF`, `WHILE`, or `WAIT_UNTIL`). For example:

```
WAIT_UNTIL(PROBE_REG & 0x09)
```

Note: The bitwise AND operator can only be used with the DSPL registers, and will **NOT** work with DSPL variables, state variables, or table values.

& cont. OPERATOR

SEE ALSO `~, AND, OR, IF, WAIT_UNTIL, WHILE`

EXAMPLE

The conditional expression used in the IF statement below masks out all bits except bits 1 and 3 of input register 2:

```
IF(INP2_REG & 0x0A)
```

<, >, <=, >=, ==, != **OPERATOR**

OPERATORS < (Less than), > (Greater than), <= (Less than or equal to)

>= (Greater than or equal to), = (Equal to), != (Not equal to)

SYNTAX `valu1 OP valu2`

USAGE DSPL (PLC, Motion)

ARGUMENTS

valu1	A DSPL variable or state variable
OP	One of the following relational operators: <, >, <=, >=, ==, !=
valu2	A DSPL variable, state variable or a constant real number.

DESCRIPTION

The relational operators are used to compare two values. A result of 1 is returned only if the specified relationship between the two values is true. otherwise a result of 0 is returned.

Note: These operators are only used in DSPL conditional statements `inside of a DSPL conditional structure (i.e. IF, WHILE, or WAIT_UNTIL)`. For example:

```
WAIT_UNTIL(VAR1 >= 1000)
```

Note: No more than one of the two values to be compared can be a state variable. The following is therefore **NOT** a valid line of DSPL code

```
IF (POS1 <= POS4)
```

SEE ALSO `~, &, AND, OR, IF, WAIT_UNTIL, WHILE`

<, >, <=, >=, ==, != cont. OPERATOR

EXAMPLE

In the first example, the `WAIT_UNTIL` statement below will stop the execution of the DSPL code as long as the actual position of axis 1 is equal to 1000:

```
WAIT_UNTIL(POS1 == 1000)
```

In the second example, the `WAIT_UNTIL` statement below will stop the execution of the DSPL code as long as the actual velocity of axis 3 is less than the value stored in `VAR25`:

```
WAIT_UNTIL(VEL3 < VAR25)
```

In the third example, the `WAIT_UNTIL` statement below will stop the execution of the DSPL code as long as the value in `VAR19` is greater than or equal to 225.7:

```
WAIT_UNTIL(VAR19 >= 225.7)
```

In the fourth example, the `WAIT_UNTIL` statement below will stop the execution of the DSPL code as long as the value stored in `VAR60` is less than or equal to the value stored in `VAR1`:

```
WAIT_UNTIL(VAR60 <= VAR1)
```