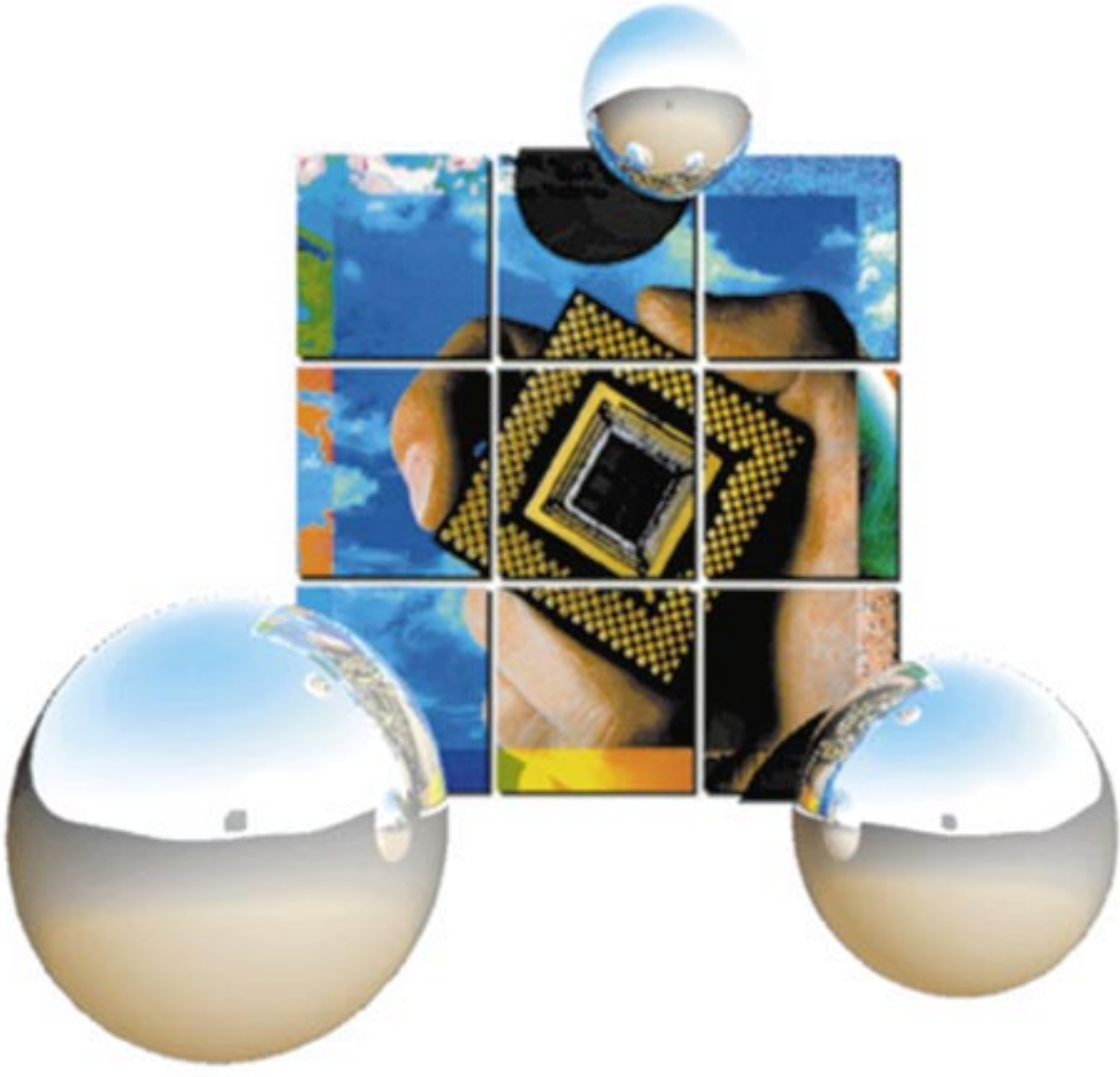
Mx4 cnC++

User's Guide v1.1



Mx4 cnC++

User's Guide

v1.1

© Copyright 1991-1995 DSP Control Group, Inc.
PO Box 39331
Minneapolis, MN 55439
Phone: (612) 831-9556
FAX: (612) 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

Read This First

Congratulations on purchasing Mx4 cnC++, DSP Control Group's high-speed multi-DSP based motion controller. You will find Mx4 cnC++ a powerful controller with an instruction set suitable for all coordinated motion control applications.

In conjunction with this manual, the following four manuals will assist you to develop and integrate Mx4 cnC++ into your simple or complex machine. Depending on your application and system integration expertise, you may find none, one or more of these supplementary manuals necessary.

Mx4Pro: Mx4 Tuning Expert

This manual describes Mx4Pro - a testing and tuning software used with Mx4 cnC++. Mx4Pro includes features such as a signal generator oscilloscope and live block diagram which make this software useful for testing and performance optimization.

Mx4 and C Programmer's Guide

This manual is written for those who wish to know about programming Mx4 in the C environment. *Mx4 & C* assumes a minimum background in C programming and in simple words describes efficient programming in the x86 environment.

Mx4 and Windows, Programming Mx4 in C++

This manual contains information on the Mx4 Windows Programming Library (DLL) which permits C and C++ Windows applications to directly interface with the Mx4 controller. Using the library applications can issue RTCs to Mx4 and obtain the values of process variables made available by the controller.

VECTOR4 User's Guide

This manual includes information on the add-on drive control option. VECTOR4 is DSPCG's multi-DSP based drive controller that provides complete drive signal

processing for all industrial DC and AC machines. The capabilities of VECTOR4 include that normally offered by servo control amplifiers.

Mx4 Development System User's Guide

This manual describes simple instruction on how DSPCG's full Mx4 development system works. The development system includes:

One Mx4 PC/AT
One VECTOR4 Add-On Drive Control Card
Three Self-Protected Power Modules for three axes of AC motor control
One Brushless DC Motor (1 hp) with 1000 line incremental encoder
One AC Induction Motor (1 hp) with 1000 line incremental encoder
One Power Cabinet
One Switching DC Power Supply
Set of Cables

A Quick Overview of This Manual

First, we would like to share with you the way this manual is organized, hoping this knowledge will help you quickly find the information you need.

We feel the first step in using an involved computerized product like Mx4 cnC++ is to understand its definition and topology (the way it is connected to other subsystems and functions) in a system. Chapter 1 is dedicated to this task. This chapter contains simple block diagrams that will describe Mx4 cnC++'s capabilities and functions. Please bear in mind that detailed information on Mx4 cnC++ is provided in the following chapters, and in Chapter 1 we only describe this product qualitatively.

Once you have learned about the basic functions of Mx4 cnC++, you may want to test Mx4 cnC++'s strength in your system. Chapter 2 provides you with information on hardware installation. In this chapter you will find information on the location of jumpers and DIP switches, wiring Mx4 cnC++ to your amplifier, I/O and encoder subsystems, Mx4 cnC++ memory space address settings, interrupt request jumper/DIP switch settings, etc.

Note: Please always read the "README" file in the root directory of the enclosed Mx4 Utilities diskette for the latest updates.

The accompanying 3.5" diskette contains a program called MX4TEST.EXE. This test program is very useful in initial hardware installation and power-up. In short you can use this program as a quick "sanity check" on your wiring and switch settings.

Once you have installed your hardware and made sure that all communications, switches and jumpers are made and set correctly, you may move onto Chapter 3. Chapter 3 briefly describes the main features of the Mx4Pro development software. *Mx4Pro* is an easy-to-use graphic program that allows for quick system programming and tuning (useful for single and multi-axis applications). No programming skills are required to use this program. When running *Mx4Pro*, all functions of Mx4 cnC++ are menu selectable; this is done to ensure you will focus on system tuning and optimization and won't be bogged down with programming details. Mx4Pro is supported by its independent manual. You should feel comfortable with all the functions of Mx4 cnC++ and tune the control gains to your satisfaction before you move onto the next chapter.

Beginning Chapter 4 and beyond, information on Mx4 cnC++ will become more technical and "lower level". You must deal with Mx4 cnC++ at this level of detail only when you are ready to write your own customized application program. Chapter 5 is dedicated to the description of Mx4 cnC++'s low-level instruction set. For each instruction we describe its function, code, arguments and a few applications that can benefit from its strength. Please remember that this chapter explains what each instruction does but not how it can be transmitted to Mx4 cnC++ by the host.

We dedicated Chapter 6 to explain Host-Mx4 cnC++ communication. To understand the communication between the host computer and Mx4 cnC++, users are required to know about Mx4 cnC++'s memory organization, the address space each Mx4 cnC++ card occupies and software communication protocols. In describing Mx4 cnC++'s memory organization, we have categorized the Mx4 cnC++ commands into two major groups of real time commands (the commands that receive the DSP's immediate attention) and contouring commands (those that are executed based on the order they are stored in a special location of the dual port memory called the ring buffer). You will also learn how Mx4 cnC++

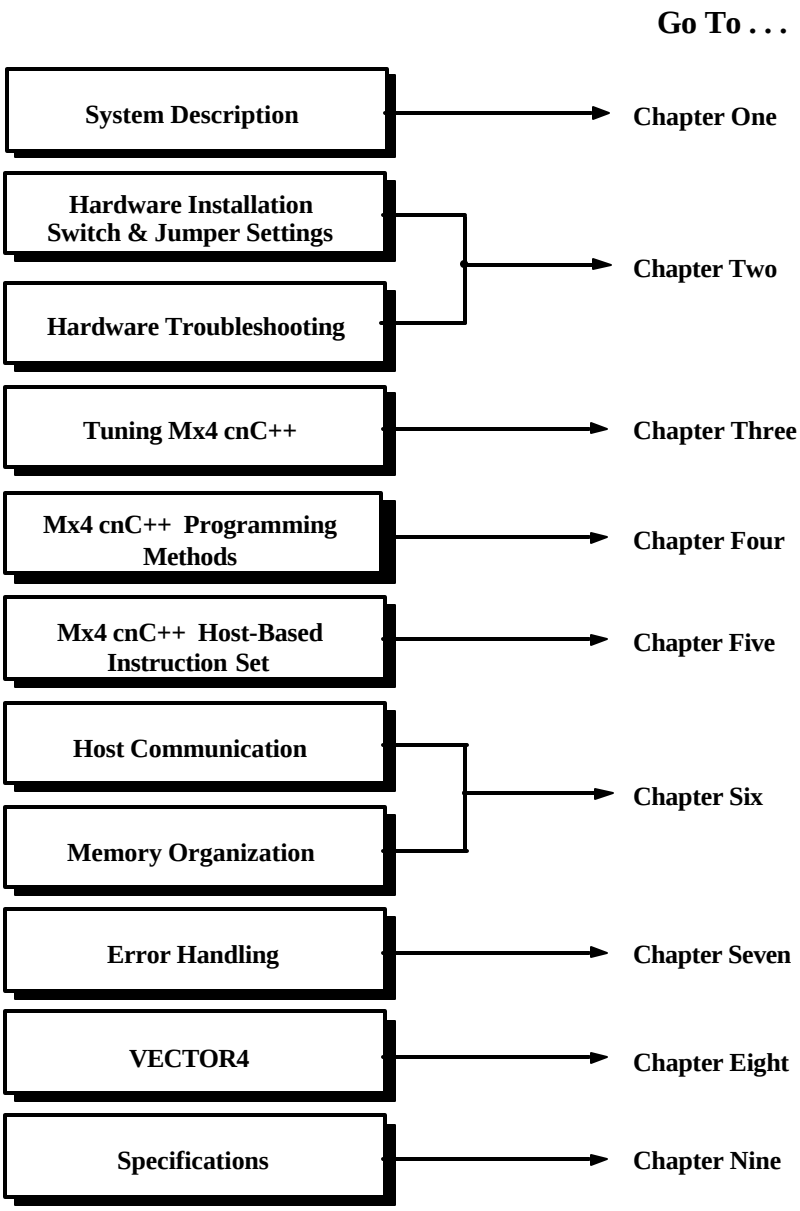
reports back to the host (special dual port memory location dedicated to these parameters) and what situations cause Mx4 cnC++ to interrupt the host.

In Chapter 7 we describe the sources of errors, how Mx4 cnC++ reports them to the host, how the user application program must handle them, and finally, possible ways they may be cured.

Chapter 8 briefly describes VECTOR4, an add-on multi-DSP based card for drive applications. If you have only purchased a Mx4 cnC++ card, you can skip this chapter.

Finally, Chapter 9 is devoted to Mx4 cnC++'s specifications. Detailed electrical and mechanical specifications are listed. On the hardware side we have included all bus specific information. Numerical values for Mx4 cnC++'s parameters and variables have been listed in terms of their binary range. Parameters specifying performance such as sampling period and the maximum encoder speed Mx4 cnC++ can handle have been listed under performance specifications.

A Quick Reference



This page intentionally blank.

1 Introduction to Mx4 cnC++

Mx4 cnC++ System Description

Mx4 cnC++ is a fully digital high-performance four-axis position controller. This multi-DSP based servo controller uses up to four DSPs (including a drive control option, VECTOR4) in a parallel processing configuration to close tighter, faster and more robust position and velocity loops. It also utilizes DSPCG's ASIC technology which provides exceptional hardware versatility and flexibility.

Mx4 cnC++ outputs its control signals (ranging ± 10 volts) via four 16-bit parallel DACs to any AC/DC industrial servo amplifier. It also incorporates 8 on-board inputs and 3 outputs for PLC applications. See Fig. 1-1.

Using VECTOR4, the optional add-on drive control card, converts the Mx4 cnC++ to a "complete signal processing unit". The complete unit is capable of performing all control functions including PWM signals for the power transistors of an output stage.

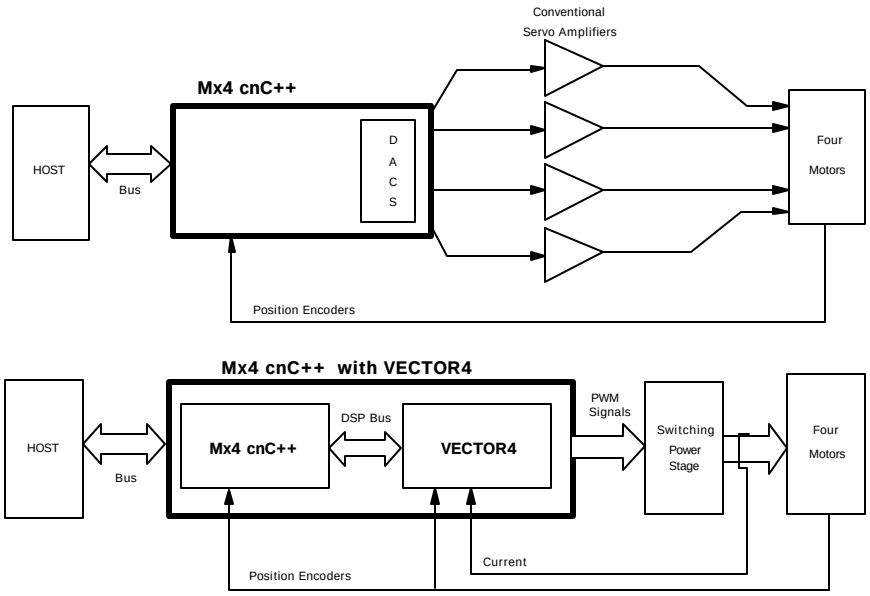


Fig. 1-1: *Top*: Mx4 cnC++ with Conventional Servo Amplifiers,
Bottom: Mx4 cnC++ with the VECTOR4 Drive Control Option

Mx4 cnC++ Programming

Mx4 cnC++ incorporates RTC programming. In addition, Mx4 cnC++ supports contouring commands for complex control applications.

The Real Time Commands, or RTCs, are issued by the host and executed by Mx4 cnC++ immediately after they are transferred. Contouring commands are issued by the host in the form of transferring a number of widely spaced position and velocity points to Mx4 cnC++. These commands are stacked up and executed by Mx4 cnC++ sequentially.

Mx4 cnC++ uses a Dual Port RAM (DPR) for communication with the host processor or computer. The DPR is partitioned into a large ring buffer for downloading host instructions to Mx4 cnC++ and a number of register "windows" for bi-directional information transfer. All system states such as position and velocity are reported in real-time to the DPR for the host to read. In addition, Mx4 cnC++ supports a debug feature that allows the host to interrogate internal Mx4 cnC++ parameters through the DPR.

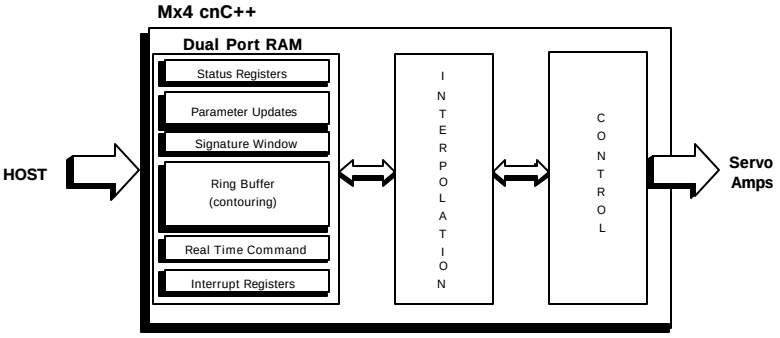


Fig. 1-2: Mx4 cnC++ Internal Functional Block

Mx4 cnC++ Computation Power & Servo Update Rate

The tremendous power of four DSPs yields 36 MIPS, million instructions per second, (four axes with 9 MIPS per axis equals 36 MIPS). This speed makes the Mx4 cnC++ the fastest controller in the world. These ultra-high speed DSPs implement advanced optimum control algorithms at a 120µs update rate (all axes included). Despite the control complexity incorporated in Mx4 cnC++, its sample rate is the fastest in the world making a 1,000 Hz analog control loop completely replaceable with the Mx4 cnC++. This leaves no practical analog servo out of the picture.

Mx4 cnC++ Control Law

The Mx4 cnC++ incorporates a state feedback controller with dual feedback loops. A single 40 MHz DSP is dedicated to this task because the control law is important in control quality

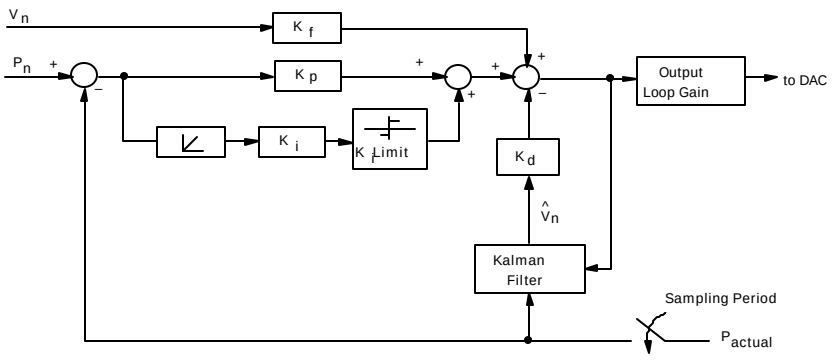


Fig. 1-3: Mx4 cnC++ Position, Velocity Control Block Diagram (excluding drive control)

This control includes position and velocity loops. The actual system speed is estimated by the Kalman filter and fed back to regulate speed. The two states, position and speed, are constantly commanded by an interpolating algorithm and maintained by the control law. The control law is closed at 120ms (all axes included), providing robust operation for all industrial applications demanding up to a 1,000 Hz position control bandwidth.

Why State Feedback?

The answer is simple: state feedback is easier to tune and provides a combination of control robustness with high bandwidth. Within this structure, optimum control algorithms such as LQG, dead beat, bang bang, etc. may be implemented.

The Kalman filter provides optimum estimation of speed and acceleration when environmental noise is present. The Kalman filter's output, velocity, provides the best feedback information at speeds with a low encoder pulse rate. The Kalman filter yields the best speed regulation at very low speeds.

In addition to state feedback control, an integration channel with anti-windup capability is provided to enable users to implement a traditional PID algorithm.

Notch Filter

Mx4 cnC++ includes a [optional] notch filter with programmable notch frequency. This feature eliminates the mechanical resonance caused by an imperfect coupling between motor and load or other joint flexibility.

Drive Control Law

Mx4 cnC++ includes a drive control option, VECTOR4, that regulates current loops. This is essential in robust and high bandwidth control of multi-phase industrial drives. Two additional DSPs are dedicated to this task.

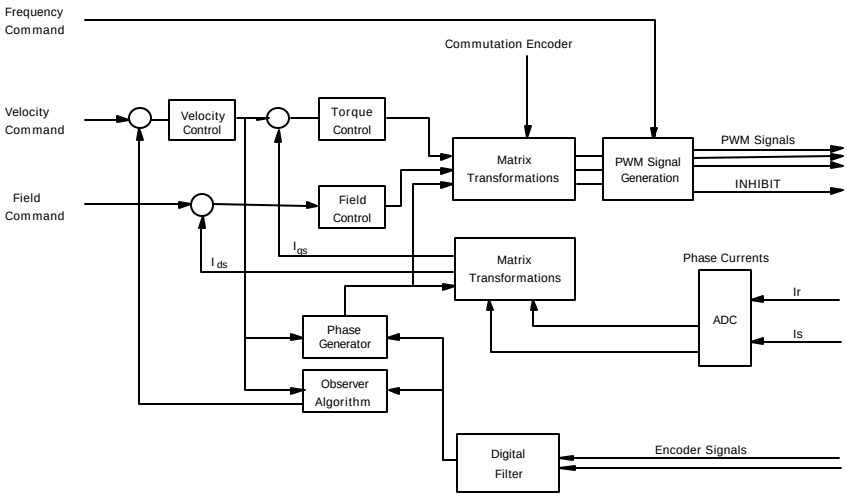


Fig. 1-4: Mx4 cnC++ Drive Control Block Diagram

Coordination

Coordination of four axes requires breaking four dimensional motion vectors down to the individual axis and interpolating the segment positions. Large position segments such as circular and elliptic arcs (for four or more axes) are broken down to smaller position and velocity pieces. These segments are interpolated down to 200 µs intervals. This provides the tight coordination ideal for CNC, robotics and other applications demanding high-speed precision control.

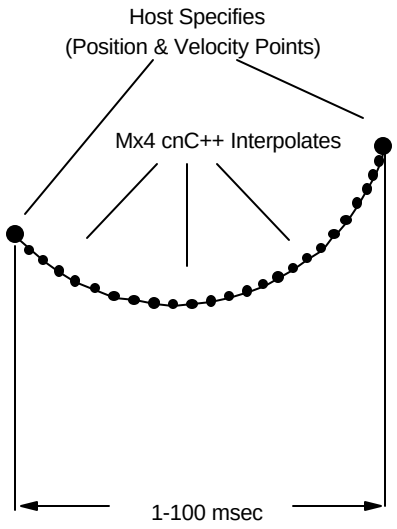


Fig. 1-5: Mx4 cnC++ Interpolation

Cubic Spline Contouring

This interpolation provides a path between two user-specified position points which is smooth in velocity and continuous in acceleration. Cubic spline interpolation enhances contouring quality especially when the position points are widely spaced in time.

Fig. 1-6 compares the linear and cubic spline interpolations. The following figure (Fig. 1-7), shows the significance of cubic spline interpolation over that of linear when first (velocity) and second (acceleration) derivatives are considered.

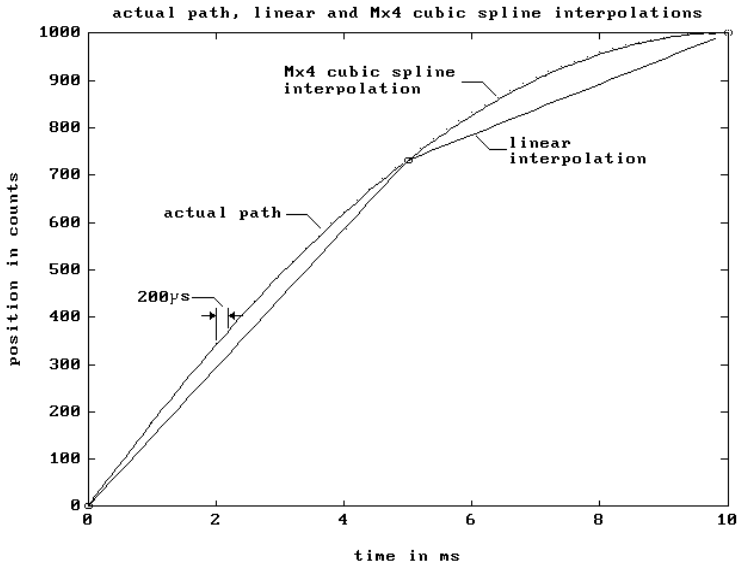


Fig. 1-6: Three User-Specified Pos./Vel. Points, Linear Interpolation and Mx4 Cubic Spline Interpolation

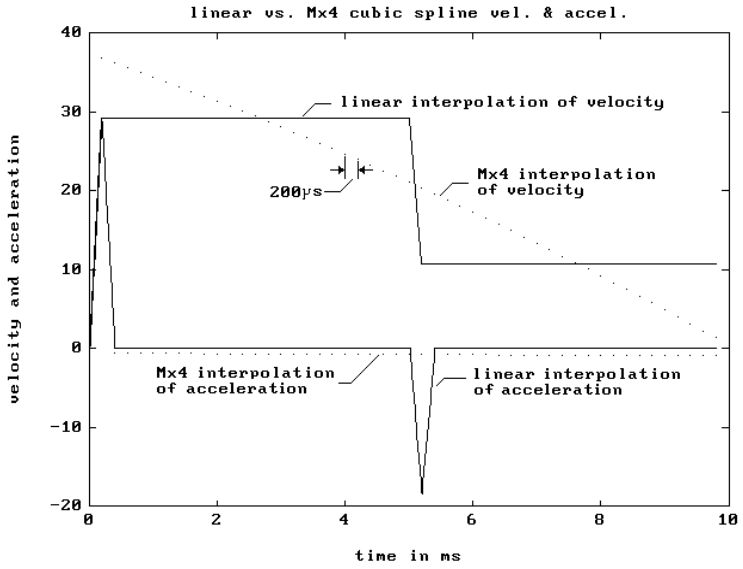


Fig. 1-7: Mx4 Cubic Spline Interpolations vs. Common Linear Interpolation

5000 blocks of position/velocity per second are transferred to four control loops.

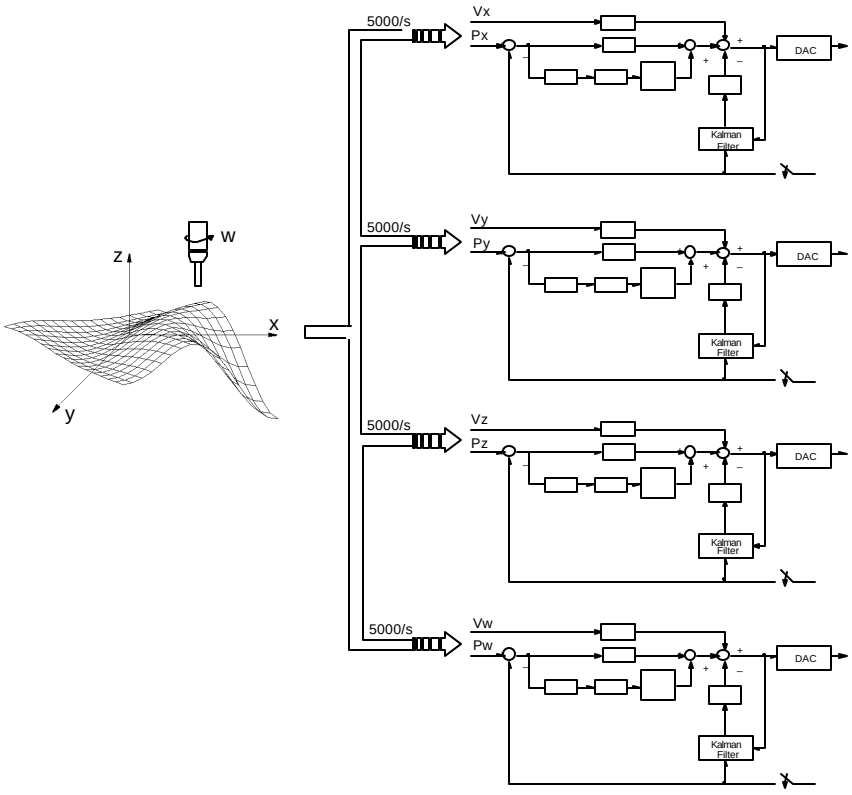


Fig. 1-8: Mx4 Cubic Spline Block Transfer

The combination of fast block transfer rate and cubic spline interpolation improve contouring speed and resolution. This is illustrated by Fig. 1-9, scope picture of x-axis position.

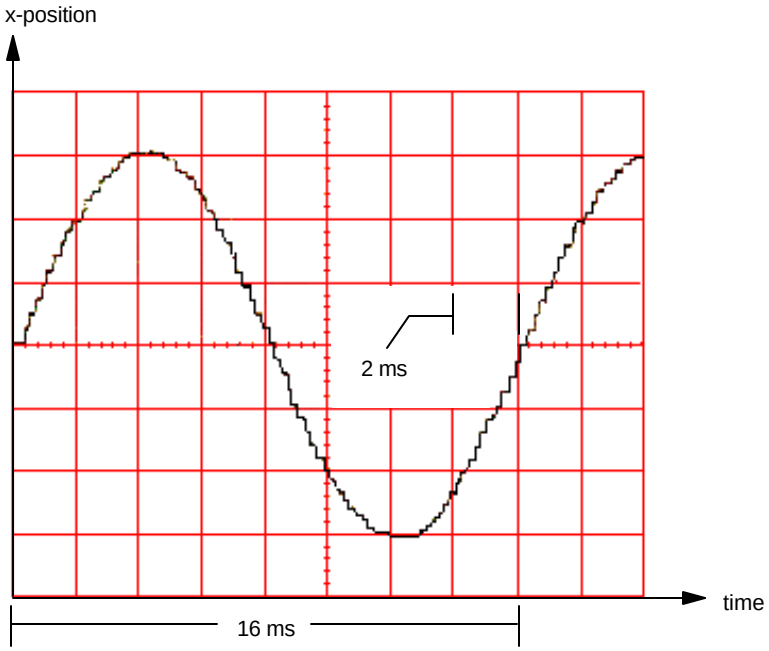


Fig. 1-9: The Mx4 DAC output for x-axis position illustrating cubic spline interpolation through 16 points

This picture shows the x-axis share of a circular move performed in 16 ms! The significance of this graph is not that the Mx4 can finish a circle in 16 ms but that it can perform sharp edge contours at high feed rates. For example, one can claim that the x-y scope plot of Fig. 1-10 is a natural benefit of Fig. 1-9.

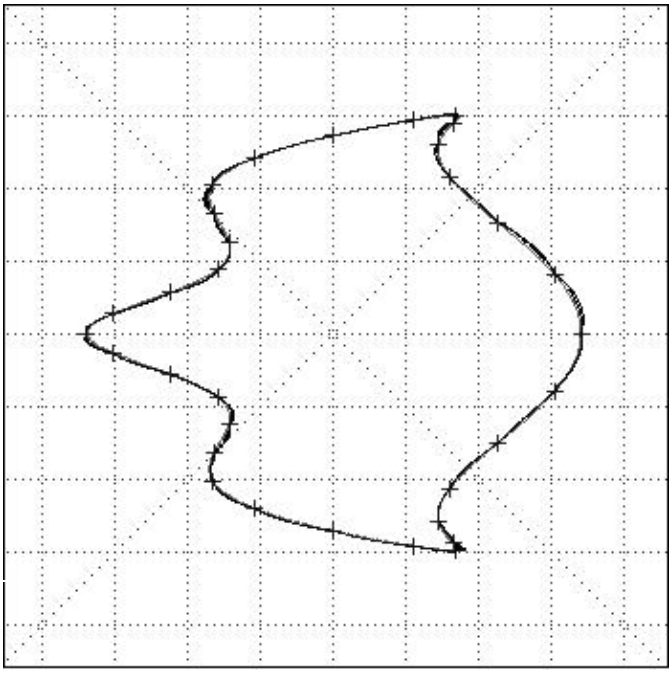


Fig. 1-10: The Mx4 x-y position plot using cubic spline contouring of 32 points.

i) Total contouring time = 160 ms,
ii) "+" marks indicate the 32 supplied points,
iii) Continuous line illustrates the Mx4's interpolated path.

Synchronization

In addition, Mx4 cnC++ synchronizes several axes of control using high-speed (100 ns) position and event captures. In applications such as printing, packaging, indexing, paper handling, etc., the initial motion in several axes depends on the position of a master axis or a timing pulse provided by an external event. Proper timing for the execution of motion is crucial for synchronized applications. The Mx4 cnC++'s ASICs contain 100 ns position and event captures designed especially for these applications.

2 Installing Your Mx4 cnC++ Hardware

A typical PC/AT Mx4 cnC++ system (Fig. 2-1) consists of:

1. a PC/AT ISA host computer
2. a Mx4 cnC++ card occupying a slot on the host computer
3. one to four motors with incremental position encoder(s)
4. one to four servo amplifiers
5. cabling from Mx4 cnC++ J6 connector to servo amplifier(s)
6. encoder feedback cabling to Mx4 cnC++ J6 connector
7. optional cabling of external inputs to Mx4 cnC++ (J3 or J6) connector
8. optional cabling of user inputs/outputs to Mx4 cnC++ J3 connector
9. optional synchronization cable between multiple Mx4 cnC++ cards

When installing a Mx4 cnC++ card, it is important to follow a procedure so that the card operates correctly in a given system. The installation guidelines detailed here incorporate three important topics: cabling to the Mx4 cnC++ card, Mx4 cnC++ jumper settings and bus-related Mx4 cnC++ settings.

Note: If you are impatient to test the communication between your computer and the Mx4 cnC++ card before completing the instructions of this chapter, you may do so by running Mx4 Pro software (see Chapter 3, *Running Mx4Pro* section). The monitor will display the MAIN MENU screen only if the Mx4 cnC++ card and your computer are communicating.

When you are assured of this communication, come back to finish this chapter!!

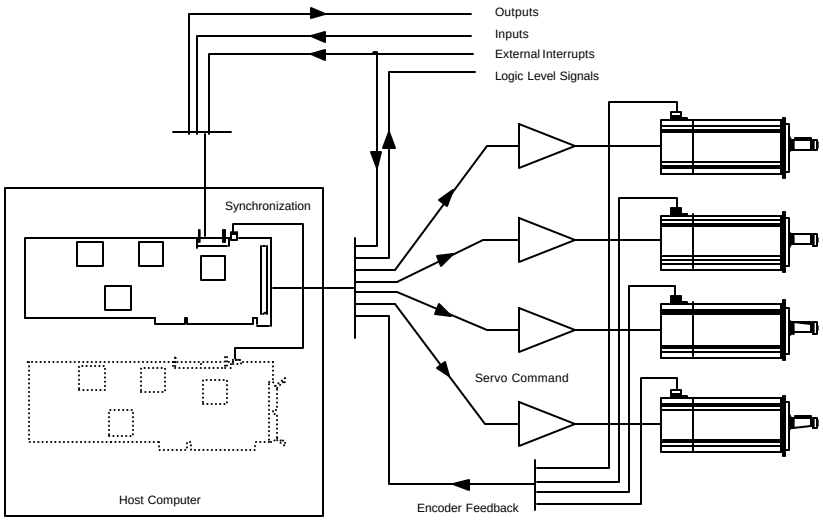


Fig. 2-1: PC/AT Mx4 cnC++ System Cable Diagram

Fig. 2-2 is an illustration of a PC/AT Mx4 cnC++ card that details connector, jumper and DIP switch positions and orientations. This figure will be used as a reference in the following pages.

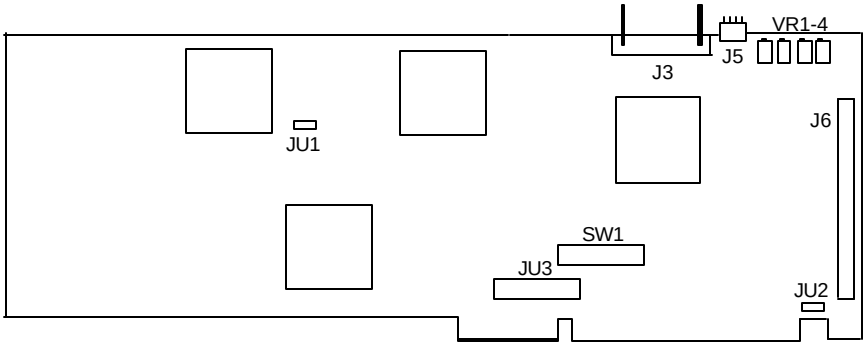


Fig. 2-2: PC/AT Mx4 cnC++ Card Component Side

PC/AT Mx4 cnC++ Cabling

The PC/AT Mx4 cnC++ card contains three connectors as illustrated in Fig. 2-2. These connectors are used for interfacing the Mx4 cnC++ card to the motors/system, optional user-defined inputs and outputs, and for the synchronization of multiple PC/AT Mx4 cnC++ cards (Fig. 2-3).

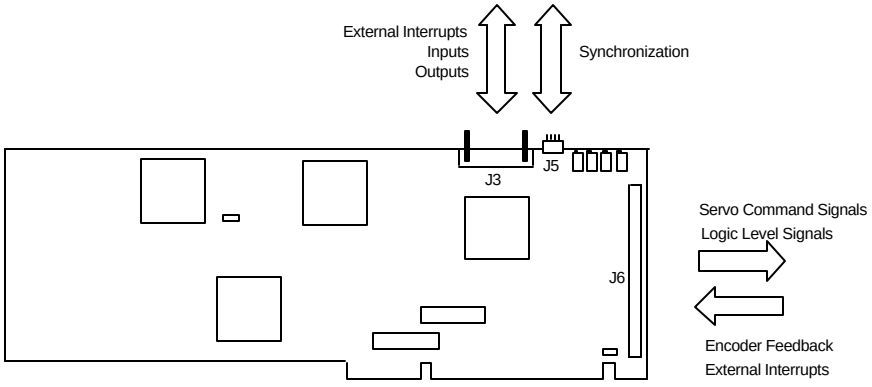


Fig. 2-3: PC/AT Connector Signals

Before using a Mx4 cnC++ card in a system application, a cable 'network(s)' must be built. The following sections provide a reference for designing and building PC/AT Mx4 cnC++ cables.

PC/AT Mx4 cnC++ J6 Connector ..... Motor/System Interfacing

The PC/AT Mx4 cnC++ J6 connector is a (50-pin dual row header). This connector includes the motor and system interfacing signals for four axes. The signals are divided into four categories: servo command signals, encoder feedback signals, general purpose external interrupt inputs and logic level signals.

Table 2-1 specifies the pinout for the PC/AT Mx4 cnC++ 50-pin header. The table includes signal level (type) and I/O functionality (with respect to the Mx4 cnC++ card).

J6 Connector Pinout

PIN	SIGNAL	LEVEL	I/O	DESCRIPTION
1	+12 volts	-	O	-
2	+5 volts	-	O	-
3	+12 volts	-	O	-
4	-12 volts	-	O	-
5	Digital GND	-	O	-
6	Analog GND	-	O	-
7	Shield GND	-	O	-
8	ESTOP/	TTL	I	Mx4 cnC++ emergency stop input
9	/PR0	TTL	I	general purpose (probe) external interrupt
10	/PR1	TTL	I	general purpose (probe) external interrupt
11	DAC(1)	-10 to +10v	O	DAC/motor output for axis 1
12	Analog GND	-	O	-
13	Digital GND	-	O	-
14	A+(1)	TTL	I	differential encoder signal A+ for axis 1
15	A-(1)	TTL	I	differential encoder signal A- for axis 1
16	B+(1)	TTL	I	differential encoder signal B+ for axis 1
17	B-(1)	TTL	I	differential encoder signal B- for axis 1
18	+5 volts	-	O	-
19	IP+(1)	TTL	I	differential encoder index pulse signal IP+ for axis 1
20	IP-(1)	TTL	I	differential encoder index pulse signal IP- for axis 1
21	DAC(2)	-10 to +10v	O	DAC/motor output for axis 2

Table 2-1: PC/AT Mx4 cnC++ J6 Connector Pinout (continued on next page)

PIN	SIGNAL	LEVEL	I/O	DESCRIPTION
22	Analog GND	-	O	-
23	Digital GND	-	O	-
24	A+(2)	TTL	I	differential encoder signal A+ for axis 2
25	A-(2)	TTL	I	differential encoder signal A- for axis 2
26	B+(2)	TTL	I	differential encoder signal B+ for axis 2
27	B-(2)	TTL	I	differential encoder signal B- for axis 2
28	+5 volts	-	O	-
29	IP+(2)	TTL	I	differential encoder index pulse signal IP+ for axis 2
30	IP-(2)	TTL	I	differential encoder index pulse signal IP- for axis 2
31	DAC(3)	-10 to +10v	O	DAC/motor output for axis 3
32	Analog GND	-	O	-
33	Digital GND	-	O	-
34	A+(3)	TTL	I	differential encoder signal A+ for axis 3
35	A-(3)	TTL	I	differential encoder signal A- for axis 3
36	B+(3)	TTL	I	differential encoder signal B+ for axis 3
37	B-(3)	TTL	I	differential encoder signal B- for axis 3
38	+5 volts	-	O	-
39	IP+(3)	TTL	I	differential encoder index pulse signal IP+ for axis 3
40	IP-(3)	TTL	I	differential encoder index pulse signal IP- for axis 3
41	DAC(4)	-10 to +10v	O	DAC/motor output for axis 4
42	Analog GND	-	O	-
43	Digital GND	-	O	-
44	A+(4)	TTL	I	differential encoder signal A+ for axis 4
45	A-(4)	TTL	I	differential encoder signal A- for axis 4
46	B+(4)	TTL	I	differential encoder signal B+ for axis 4
47	B-(4)	TTL	I	differential encoder signal B- for axis 4
48	+5 volts	-	O	-
49	IP+(4)	TTL	I	differential encoder index pulse signal IP+ for axis 4
50	IP-(4)	TTL	I	differential encoder index pulse signal IP- for axis 4

Table 2-1 cont.: PC/AT Mx4 cnC++ J6 Connector Pinout

Servo Command Signals

The servo command signals are those signals that 'drive' the axis servo amplifiers or output stage. The PC/AT Mx4 cnC++ card utilizes 16-bit DAC outputs with +10v to -10v voltage swings to drive any voltage level sensitive output stage. The PC/AT Mx4 cnC++ servo command signals are listed in Table 2-2:

Partial J6 Connector Pinout

SIGNAL	PIN	LEVEL	I/O	DESCRIPTION
DAC(1)	11	-10 to +10v	O	DAC/motor output for axis 1
DAC(2)	21	-10 to +10v	O	DAC/motor output for axis 2
DAC(3)	31	-10 to +10v	O	DAC/motor output for axis 3
DAC(4)	41	-10 to +10v	O	DAC/motor output for axis 4
Analog GND	12	-	O	-
Analog GND	22	-	O	-
Analog GND	32	-	O	-
Analog GND	42	-	O	-

Table 2-2: PC/AT Mx4 cnC++ J6 Servo Command Signals

The DAC(x) signals must be routed from the J6 50-pin header connector to the respective output stage servo drives. The Mx4 cnC++ Analog GND signals are included as a voltage reference for the DAC(x) signals. Analog GND should be utilized accordingly in the cabling between Mx4 cnC++ and the output stages.

DAC(x) output offset voltage may be adjusted with the VRx multi-turn potentiometer (VR1 - DAC(1), VR2 - DAC(2), etc.) The Mx4 cnC++ is shipped from the factory with minimized offset voltage.

Encoder Feedback

The Mx4 cnC++ card requires the use of incremental position encoders for motor-Mx4 cnC++ feedback. ***No*** velocity feedback (such as a tachometer) is required as Mx4 cnC++ incorporates a Kalman velocity observer algorithm. The PC/AT Mx4 cnC++ encoder feedback signals are listed in Table 2-3.

Partial J6 Connector Pinout

SIGNAL	PIN	LEVEL	I/O	DESCRIPTION
A+(1)	14	TTL	I	differential encoder signal A+ for axis 1
A-(1)	15	TTL	I	differential encoder signal A- for axis 1
B+(1)	16	TTL	I	differential encoder signal B+ for axis 1
B-(1)	17	TTL	I	differential encoder signal B- for axis 1
IP+(1)	19	TTL	I	differential encoder index pulse signal IP+ for axis 1
IP-(1)	20	TTL	I	differential encoder index pulse signal IP- for axis 1
A+(2)	24	TTL	I	differential encoder signal A+ for axis 2
A-(2)	25	TTL	I	differential encoder signal A- for axis 2
B+(2)	26	TTL	I	differential encoder signal B+ for axis 2
B-(2)	27	TTL	I	differential encoder signal B- for axis 2
IP+(2)	29	TTL	I	differential encoder index pulse signal IP+ for axis 2
IP-(2)	30	TTL	I	differential encoder index pulse signal IP- for axis 2
A+(3)	34	TTL	I	differential encoder signal A+ for axis 3
A-(3)	35	TTL	I	differential encoder signal A- for axis 3
B+(3)	36	TTL	I	differential encoder signal B+ for axis 3
B-(3)	37	TTL	I	differential encoder signal B- for axis 3
IP+(3)	39	TTL	I	differential encoder index pulse signal IP+ for axis 3
IP-(3)	40	TTL	I	differential encoder index pulse signal IP+ for axis 3
A+(4)	44	TTL	I	differential encoder signal A+ for axis 4
A-(4)	45	TTL	I	differential encoder signal A- for axis 4
B+(4)	46	TTL	I	differential encoder signal B+ for axis 4
B-(4)	47	TTL	I	differential encoder signal B- for axis 4
IP+(4)	49	TTL	I	differential encoder index pulse signal IP+ for axis 4

IP-(4)	50	TTL	I	differential encoder index pulse signal IP+ for axis 4
Digital GND	13	-	O	-
Digital GND	23	-	O	-
Digital GND	33	-	O	-
Digital GND	43	-	O	-

Table 2-3: PC/AT Mx4 cnC++ J6 Encoder Feedback Signals

The PC/AT Mx4 cnC++ allows the use of either differential or single-ended encoder feedback. The choice is made via the JU2 jumper on the Mx4 cnC++ card (see *PC/AT Mx4 cnC++ Jumper Settings*). If a combination of differential and single-ended encoder feedback is desired, the jumper must be placed in "differential" mode and the following procedure must be followed. If single-ended encoders are to be used in "differential" mode, it is necessary to route the single-ended line to the corresponding "+" Mx4 cnC++ differential input. The "-" differential input to the corresponding signal must be tied to +2.5v. For example, to connect a single-ended "A" encoder line to axis 3 of Mx4 cnC++:

A+(3) = single-ended "A" encoder line
A-(3) = +2.5v

The Mx4 cnC++ encoder feedback inputs are TTL-level inputs. The Mx4 cnC++ Digital GND signals are included as voltage references for the differential inputs. The Digital GND signal(s) available on the J6 connector must be connected to the appropriate incremental encoder voltage reference points.

When interfacing incremental encoders to Mx4 cnC++, it is important that the following two conventions are followed:

1. Mx4 cnC++ detects an active-HIGH index pule. If the encoder(s) being interfaced to Mx4 cnC++ include index pulse signals, it is important to note that the correct polarity is in effect. To reverse the polarity of an index pulse signal, simply 'swap' the IP+ and IP- signals to the Mx4 cnC++ card.

2. The incremental encoder signals (A+, A-, B+, B-) should follow the convention of Fig. 2-4. That is, when the motor shaft is manually turned in the clockwise direction, a negative velocity should result.

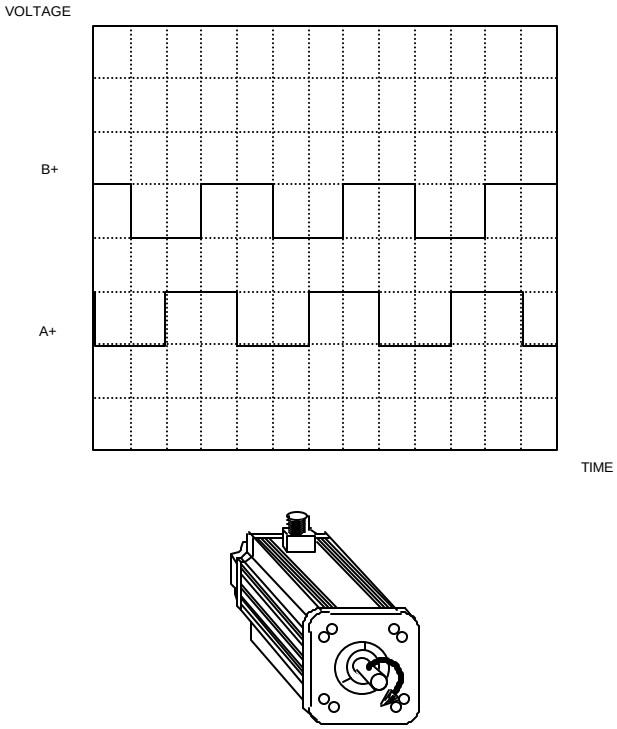


Fig. 2-4: Mx4 cnC++ Incremental Encoder Signals Polarity ... Clockwise Shaft Rotation Yields Negative Velocity

If the use of an oscilloscope is not convenient, the encoder signal polarity may be verified later. In a following section, *Verifying Your Mx4 cnC++ Hardware Set-Up*, the encoder signal polarity is checked via software. The check is simple and does not require the use of an oscilloscope. If the incremental encoder signal polarity is incorrect, it may be reversed simply by 'swapping' the A and B encoder signals.

General Purpose External Interrupt Inputs

The PC/AT Mx4 cnC++ external interrupt inputs include an 'Emergency Stop' line and two general purpose external interrupts. The J6 connector external interrupt inputs are repeated on the J3 connector. If the user is utilizing these signals, the signals may be accessed from either the J6 or J3 connector, but not from both. The external interrupt inputs are listed in Table 2-4:

Partial J6 Connector Pinout

SIGNAL	PIN	LEVEL	I/O	DESCRIPTION
ESTOP/	8	TTL	I	Mx4 cnC++ emergency stop
/PR0	9	TTL	I	general purpose (probe) external interrupt
/PR1	10	TTL	I	general purpose (probe) external interrupt
Digital GND	5	-	O	-

Table 2-4: PC/AT Mx4 cnC++ J6 External Inputs

These signals are optional for Mx4 cnC++ operation. The definitions of these signals will be presented in later sections of this manual.

The ESTOP/ and /PRx signals are active-LOW signals. That is, Mx4 cnC++ detects these active conditions when the voltage level on those lines is LOW. The Mx4 cnC++ Digital GND signal is included as a voltage reference for the external input signals.

Logic Level Voltages / GND Signals

The PC/AT Mx4 cnC++ card includes in its connector pin-out the following logic level / GND signals (Table 2-5) [in addition to the previously mentioned logic signals included with different signal groups]:

Partial J6 Connector Pinout

SIGNAL	PIN	LEVEL	I/O	DESCRIPTION
+12 volts	1	-	O	-
+12 volts	3	-	O	-
+5 volts	2	-	O	-
-12 volts	4	-	O	-
Analog GND	6	-	O	-
Digital GND	5	-	O	-
Shield Gnd	7	-	O	-

Table 2-5: PC/AT Mx4 cnC++ J6 Logic Level / GND Signals

PC/AT Mx4 cnC++ J3 Connector ... Inputs/Outputs

The PC/AT Mx4 cnC++ J3 connector is a (16-pin dual row header). This connector includes the Mx4 cnC++ input and output signals as well as a repeat of the J6 general purpose external interrupt inputs.

Table 2-6 specifies the pinout for the PC/AT Mx4 cnC++ 16-pin header. The table includes signal level (type) and I/O functionality (with respect to the Mx4 cnC++ card).

J3 Connector Pinout

PIN	SIGNAL	LEVEL	I/O	DESCRIPTION
1	OUT3	TTL	0	Gen. purpose output
2	-O.T.	TTL	I	Axis 4
3	OUT2	TTL	0	Gen. purpose output
4	+O.T.	TTL	I	Axis 4
5	OUT1	TTL	0	Gen. purpose output
6	-O.T.	TTL	I	Axis 3
7	IN5	TTL	I	general purpose input
8	+O.T.	TTL	I	Axis 3
9	IN4	TTL	I	general purpose input
10	-O.T.	TTL	I	Axis 2
11	IN3	TTL	I	Dedicated Input (ESTOP/)
12	+O.T.	TTL	I	Axis 2
13	IN2	TTL	I	Dedicated input (/PR1)
14	-O.T.	TTL	I	Axis 1
15	IN1	TTL	I	Dedicated input (/PRO)
16	+O.T.	TTL	I	Axis 1

Table 2-6: PC/AT Mx4 cnC++ J3 Connector Pinout (continued on next page)

Inputs

Mx4 cnC++ includes 10 user-defined TTL logic inputs. The input signals are listed in Table 2-7.

Partial J3 Connector Pinout

PIN	SIGNAL	LEVEL	I/O	DESCRIPTION
2	-O.T.	TTL	I	Axis 4
4	+O.T.	TTL	I	Axis 4
6	-O.T.	TTL	I	Axis 3
7	IN5	TTL	I	general purpose input
8	+O.T.	TTL	I	Axis 3
9	IN4	TTL	I	general purpose input
10	-O.T.	TTL	I	Axis 2
12	+O.T.	TTL	I	Axis 2
14	-O.T.	TTL	I	Axis 1
16	+O.T.	TTL	I	Axis 1

Table 2-7: PC/AT Mx4 cnC++ J3 Input Signals

The Mx4 cnC++ user-defined input signals are TTL logic level inputs. The inputs are equipped with pull-up resistors which are implemented as current sources (see Fig. 2-5).

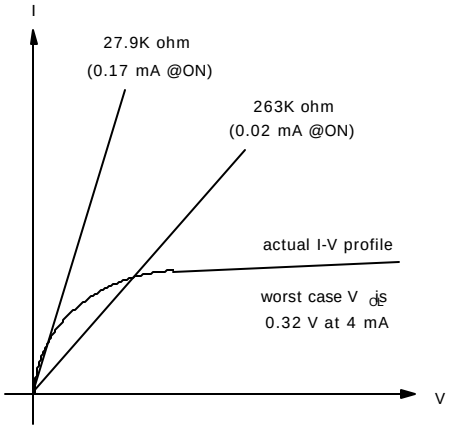


Fig. 2-5: Mx4 cnC++ Input (Pull-Up Resistor) Current Source

By default, the inputs are defined as active-LOW. That is, 0v applied to an input results in an active, or ON, input; +5v applied to an input results in an inactive, or OFF input. The logic state of the inputs may be individually selected via the INPSTATE command.

Fig. 2-6 illustrates two possible configurations for interfacing external input circuitry to Mx4 cnC++ inputs: optically-isolated input and same-ground input.

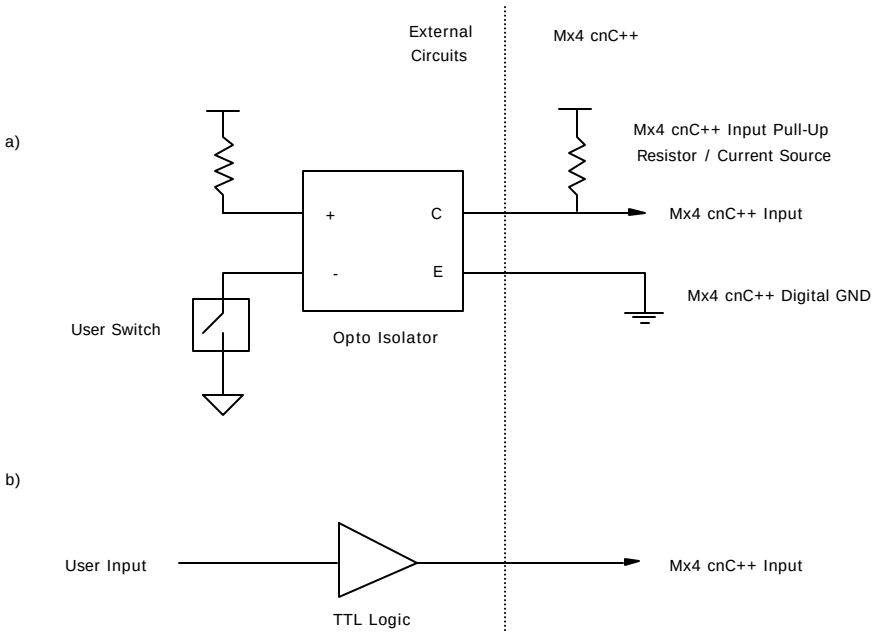


Fig. 2-6: Interfacing Input Signals to Mx4 cnC++
a) Optical Isolated Input
b) Same-Ground Input

Outputs

The PC/AT Mx4 cnC++ controller includes 3 programmable outputs. The output signals are listed in Table 2-8.

Partial J3 Connector Pinout

PIN	SIGNAL	LEVEL	I/O	DESCRIPTION
1	OUT3	TTL	O	general purpose output
3	OUT2	TTL	0	general purpose output
5	OUT1	TTL	O	general purpose output

Table 2-8: PC/AT Mx4 cnC++ J3 Output Signals

The Mx4 cnC++ output signals are TTL logic level outputs with a fan out of one (that is, a Mx4 cnC++ output should not be used to drive more than one TTL logic gate). As an example of interfacing to the Mx4 cnC++ output signals, Fig. 2-7 illustrates a relay output circuit.

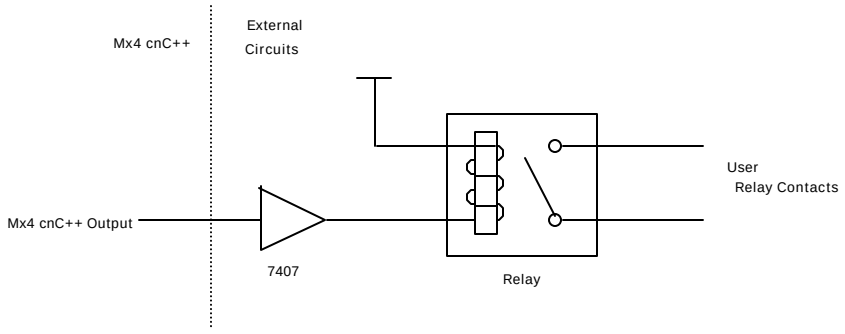


Fig. 2-7: Interfacing a Relay to a Mx4 cnC++ Output

The Mx4 cnC++ outputs are active-LOW. That is, an ON output is an output at 0v, an OFF output is an output at +5v. The ON/OFF state of the outputs is determined by the OUTREL command.

General Purpose External Interrupt Inputs

The J3 connector includes three external interrupt inputs which are repeated on the J6 connector. If the user is utilizing these signals, the signals may be accessed from either the J6 or J3 connector, but not from both. The PC/AT Mx4 cnC++ external interrupt inputs include an 'Emergency Stop' line and two general purpose external interrupts. The external interrupt inputs are listed in Table 2-9:

Partial J3 Connector Pinout

SIGNAL	PIN	LEVEL	I/O	DESCRIPTION
ESTOP/	11	TTL	I	Mx4 cnC++ emergency stop
/PR0	15	TTL	I	general purpose (probe) external interrupt
/PR1	13	TTL	I	general purpose (probe) external interrupt

Table 2-9: PC/AT Mx4 cnC++ J3 External Inputs

These signals are optional for Mx4 cnC++ operation. The definitions of these signals will be presented in later sections of this manual.

The ESTOP/ and /PRx signals are active-LOW signals. That is, Mx4 cnC++ detects these active conditions when the voltage level on those lines is LOW. The Mx4 cnC++ Digital GND signal is included as a voltage reference for the external input signals.

PC/AT Mx4 cnC++ J5 Connector ... Synchronization

Multiple PC/AT Mx4 cnC++ cards may be time-synchronized to the same DSP cycle with the J5 synchronization connector. This Mx4 cnC++ feature allows multi-axis systems which require greater than four axes to be synchronized. The Mx4 cnC++ synchronization signals are listed in Table 2-11.

J5 Connector Pinout

PIN	SIGNAL	LEVEL	I/O	DESCRIPTION
1	nc	-	-	no connection
2	SLAVE	TTL	I	slave Mx4 cnC++ synchronization input
3	SLAVE	TTL	I	slave Mx4 cnC++ synchronization input
4	MASTER	TTL	O	master Mx4 cnC++ synchronization output

Table 2-11: PC/AT Mx4 cnC++ J5 Connector Pinout

Synchronizing multiple Mx4 cnC++ cards only requires cabling between the MASTER J5 signal from the "master" Mx4 cnC++ to a SLAVE J5 signal (either pin 2 or pin 3) on the "slave" card(s).

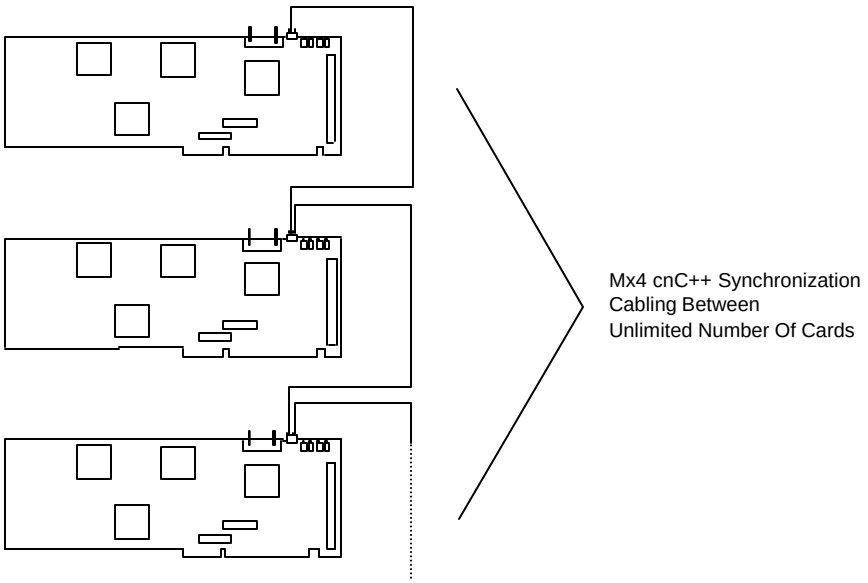


Fig. 2-8: Time Synchronizing Multiple PC/AT Mx4 cnC++ Cards

The Mx4 cnC++ J5 connector includes dual SLAVE signals in order to simplify "daisy chaining" between multiple Mx4 cnC++ controllers.

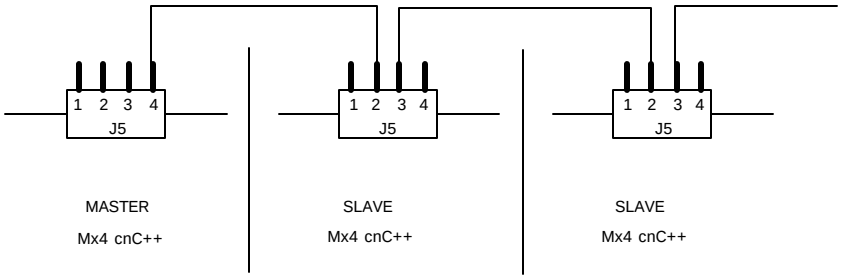


Fig. 2-9: Mx4 cnC++ J5 Connector Daisy Chaining Cabling

PC/AT Mx4 cnC++ Jumper Settings

The PC/AT Mx4 cnC++ card contains three jumpers. The jumpers should be set according to the following table. The jumper orientation on the PC/AT Mx4 cnC++ card was shown in Fig. 2-2. The three jumpers are listed below in Table 2-12.

JUMPER	# POS.	JUMPER ORIENTATION / DESCRIPTION
JU1	3	jumper must be placed in 1-2 position
JU2	3	1-2: Differential encoder operation 2-3: Single-ended encoder operation
JU3	22	interrupt selection jumper, see *PC/AT Mx4 cnC++ Bus Specifications / Settings*

Table 2-12: PC/AT Mx4 cnC++ Jumpers

PC/AT Mx4 cnC++ Bus Specifications / Settings

In order for the Mx4 cnC++ card to operate correctly on the host bus (and thus in the system), the host must be able to address the Mx4 cnC++ card and receive interrupts from the Mx4 cnC++ card. These are host computer/bus issues that require proper settings on the Mx4 cnC++ card as well as correct software routines on the host computer end. The following is a description of how the PC/AT Mx4 cnC++ card operates on the PC/AT ISA bus as well as outlines for setting the PC/AT Mx4 cnC++ bus interface parameters.

Note: The software included with the Mx4 cnC++ card (*Mx4 Test*) requires specific bus-related settings on the Mx4 cnC++ card in order to run. That is, these programs require Mx4 cnC++ to be located (address-wise) in a unique location in the host-bus address space. The bus-related Mx4 cnC++ settings are included in this chapter, *Verifying Your Mx4 cnC++ Hardware Set-Up*.

The PC/AT Mx4 cnC++ acts as a memory device on the ISA bus. It decodes 20 address bits and communicates via 8-bit data transfers. The PC/AT Mx4 cnC++ contains a jumper to select any one of 11 bus interrupt lines.

Memory Address

The PC/AT ISA bus Mx4 cnC++ decodes 20 address bits. The card can be positioned on any 2K boundary within the ISA base 1M address space. The PC/AT Mx4 cnC++ will not respond to any addresses in the upper 15M of address space. The PC/AT Mx4 cnC++ 10-position DIP switch SW1 is used to select the start of the boundary (Table 2-13).

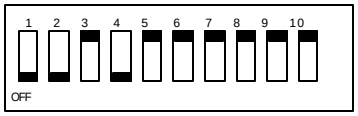
POSITION	ADDRESS MATCHED	
	SW OFF	SW ON
SW1-1	A19=1	A19=0
SW1-2	A18=1	A18=0
SW1-3	A17=1	A17=0
SW1-4	A16=1	A16=0
SW1-5	A15=1	A15=0
SW1-6	A14=1	A14=0
SW1-7	A13=1	A13=0
SW1-8	A12=1	A12=0
SW1-9	A11=1	A11=0
SW1-10	nc	nc

Table 2-13: PC/AT Mx4 cnC++ 2K Boundary Select

Example: A PC/AT Mx4 cnC++ card is to be installed into the host bus address space at the start of the 0xd, 64K segment.

The SW1 DIP switch is set as follows:

The SW1 DIP switch is set as follows:



Memory Space Functionality

The entire 2K memory space required by the PC/AT Mx4 cnC++ card is for accessing the Mx4 cnC++ 2K DPR.

Interrupt Setting

The PC/AT Mx4 cnC++ card supports 11 PC/AT ISA bus interrupt lines which are jumper selectable on the Mx4 cnC++ card (only one jumper is permitted). Mx4 cnC++ will use the selected interrupt line to signal interrupts to the host. The PC/AT Mx4 cnC++ jumper JU3 is partitioned as follows in Table 2-14 (left to right, 1-11).

JUMPER POSITION	INTERRUPT SELECTED
JU3-1	IRQ15
JU3-2	IRQ14
JU3-3	IRQ12
JU3-4	IRQ11
JU3-5	IRQ10
JU3-6	IRQ9
JU3-7	IRQ7
JU3-8	IRQ6
JU3-9	IRQ5
JU3-10	IRQ4
JU3-11	IRQ3

Table 2-14: PC/AT Mx4 cnC++ Host Bus Interrupt Select

Verifying the Mx4 cnC++ Hardware Set-Up

Important: The included *Mx4Test* software is written for PC based systems running the DOS operating system. The software requires a minimum of a 80286 processor and a VGA monitor.

The hardware installation of a PC/AT Mx4 cnC++ into a system may be verified with the use of the *Mx4Test* software that is located on the Mx4 Utilities diskette in the MX4 TEST sub directory. *Mx4Test* is an executable program that allows the user to progress through a series of tests that help determine whether or not Mx4 cnC++ is installed correctly.

Before using *Mx4Test*, it is important that the PC/AT Mx4 cnC++ bus platform-specific DIP switch SW1 is set according to the following table (Table 2-15). [*Mx4Test* requires the PC/AT Mx4 cnC++ to be located at the start of segment 0xd of the first 1M of address space.]

DIP SWITCH POSITION	SWITCH STATUS
SW1-1	OFF
SW1-2	OFF
SW1-3	ON
SW1-4	OFF
SW1-5	ON
SW1-6	ON
SW1-7	ON
SW1-8	ON
SW1-9	ON
SW1-10	not used

Table 2-15: PC/AT Mx4 cnC++ SW1 DIP Switch Setting for *Mx4Test*

Running the *Mx4Test* Software

Important: Before powering-up Mx4 cnC++ and continuing on with *Mx4Test*, the user should have followed the Mx4 cnC++ installation guidelines presented in the previous sections and set the Mx4 cnC++ switches as detailed above.

The *Mx4Test* program incorporates a variety of tests that indicate the correctness of the Mx4 cnC++ installation. Some of the tests are passive while others require action on the user's part (for example, turning a motor shaft or manually generating an external interrupt).

The *Mx4Test* tests are categorized as follows:

Mx4 cnC++ Addressing	Tests Mx4 cnC++ - host computer communication / interface.
Incremental Position	Encoder feedback to Mx4 cnC++ and feedback polarity checked. Encoder index pulse (marker) may be tested as well.
Servo Command Signals	Mx4 cnC++ digital-to-analog converter outputs are tested.
External Interrupt	Optional Mx4 cnC++ external interrupt inputs such as emergency stop and external interrupts /PR0 and /PR1 may be checked.

MX4TEST.EXE is included on the enclosed diskette in the MX4TEST directory. The program may be executed from the diskette or transferred onto a hard drive and run from there.

Once the Mx4 cnC++ card has been placed into a host-bus slot and the connector(s) is in place, the host computer (bus) may be powered-up. To run *Mx4Test* from the floppy, simply type **Mx4Test** at the a:\MX4TEST\ DOS prompt.
The *Mx4Test* Main Menu selections reflect the four categories of tests (as previously detailed). The test procedures are simple and are explained in the *Mx4Test* program.

If an error in the installation of the Mx4 cnC++ card becomes evident from *Mx4 Test*, it is advised to consult the respective section in the *Installing Mx4 cnC++ Into Your System* chapter. Some of the more common problems and related corrections are included in Table 2-16.

ITEM	POSSIBLE PROBLEM	CORRECTIVE ACTION SUGGESTIONS
Check Mx4 cnC++ Addressing	- Failure indicated by *Mx4 cnC++Test*	1. Mx4 cnC++ bus-specific DIP switches must be set according to the settings specified in *Verifying Your Mx4 cnC++ Hardware Setup* when running *Mx4 cnC++Test*
Test Incremental Position Encoders - Verify Position encoder feedback and polarity	- Position value does not change as the shaft is turned	1. Verify that encoder is powered and that Mx4 cnC++ digital GND is connected to encoder GND (voltage reference). 2. Verify proper encoder signal connections to Mx4 cnC++ as detailed in previous chapter.
	- The position polarity is reversed	1. Polarity may be reversed by "swapping" the A and B incremental encoder signals.
- Check position encoder index pulse	- No index pulse detected by Mx4 cnC++	1. Verify that encoder is powered and that Mx4 cnC++ is powered and that Mx4 cnC++ digital GND is connected to encoder GND (voltage reference). 2. Verify proper encoder index pulse signal connections to Mx4 cnC++ as detailed in previous chapter.
	- Index pulse always active, regardless of shaft position	1. Index pulse polarity may be incorrect (Mx4 cnC++ detects on active-HIGH index pulse). Polarity may be reversed by "swapping" IP+ and IP- signals to the Mx4 cnC++ card.

Table 2-16: Troubleshooting Mx4 cnC++ Installation (continued on next page)

ITEM	POSSIBLE PROBLEM	CORRECTIVE ACTION SUGGESTIONS
Test Servo Command DAC Signals	- Measured DAC voltages not correct	1. Verify that Mx4 cnC++ Analog GND is used as the voltage reference. 2. When being measured, the Mx4 cnC++ DAC output should be disconnected from the servo amplifier or output stage.
Check External Inputs - Check emergency stop input ESTOP/	- No ESTOP/ detected by Mx4 cnC++	1.ESTOP/ source must be connected to Mx4 cnC++ digital GND reference. 2. Verify ESTOP/ connection to Mx4 cnC++ connector.
	- ESTOP/ is always active	1. ESTOP/ polarity may be incorrect (Mx4 cnC++ detects an active-LOW ESTOP/.
- Check external interrupts (/PR0 and /PR1)	- /PRx interrupt not detected by Mx4 cnC++	1. Verify that /PRx source voltage reference (GND) is connected to Mx4 cnC++ digital GND. 2. Verify /PRx connection to Mx4 cnC++ connector.
	- *EXT interrupt always active	1. /PRx polarity may be incorrect (Mx4 cnC++ detects an active-LOW /PRx interrupt).

Table 2-16 cont.: Troubleshooting Mx4 cnC++ Installation

This page intentionally blank.

3 Mx4Pro Software

Important: The *Mx4Pro* software is written for PC-based systems running the DOS operating system. The software requires a minimum of a 80286 processor and a VGA monitor.

This chapter briefly overviews the features of Mx4Pro testing and tuning software. For detailed information on these features, please refer to the *Mx4Pro: Tuning Expert* manual.

Mx4 alone controls all servo amplifiers. VECTOR4 is an all-digital AC servo controller add-on card that enables Mx4 to control any combination of brushless DC, AC induction and brush-type DC motors. Fig. 3-1 illustrates the two Mx4 configurations:

a) Mx4 with traditional servo amplifiers, and
b) Mx4 with add-on card VECTOR4 and generic switching power stage

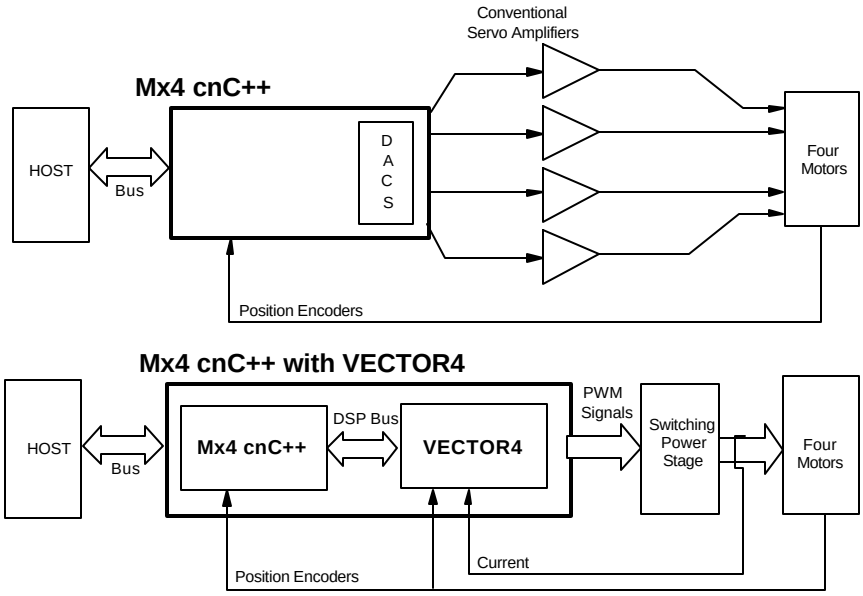


Fig. 3-1: *Top*: Mx4 cnC++ with Conventional Servo Amplifiers,
Bottom: Mx4 cnC++ with VECTOR4 drive control option

Mx4Pro is inclusive of both Mx4 cnC++ and Mx4 cnC++ with VECTOR4 functions and features. The *Mx4Pro* software package along with Mx4 cnC++ and VECTOR4 provide a powerful system. This combination allows you to customize the control to almost any combination of motor, encoder and power technologies, and tune a system for optimum performance.

In this manual, we are concerned with Mx4 cnC++ and the features of *Mx4Pro* related to the operation of a traditional servo amplifier Mx4 cnC++ system (Fig. 3-1, above). In review, Mx4 cnC++ provides the position control for four coordinated axes. The Mx4 cncC++ outputs (four 16-bit ± 10 volt DACs) must be applied to the inputs of four (current or velocity loop) servo amplifiers.

Running Mx4Pro

Important: Before powering-up Mx4 cnC++ and continuing on with *Mx4Pro*, the user should have completed the verification of their hardware set-up as detailed in the *Mx4 cnC++ User's Guide*.

MX4PRO.EXE is included on the enclosed Mx4 Utilities diskette in the root directory. The program may be executed from the diskette or transferred onto a hard drive and run from there. To run *Mx4Pro* from the floppy, simply type *Mx4Pro* at the a:\ DOS prompt. The first *Mx4Pro* screen is shown in Fig. 3-2.

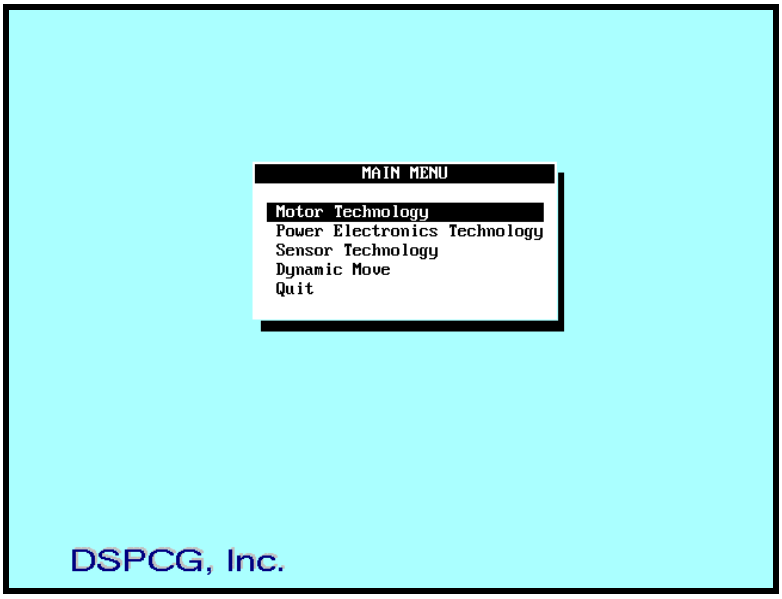


Fig. 3-2: *Mx4Pro* Main Menu Screen

Using the Keyboard with Mx4Pro

Before you proceed to the next step, we would like to share with you some key strokes required in proceeding through the *Mx4Pro* program. With *Mx4Pro*, you

only need to enter numbers representing parameter values, there is no need for text entry. However, in order to move the cursor from the current position to a new position or to select an item you need to know about a few keys.

Arrows	The *Arrow* keys are used to move up and down on a menu. *Arrow* keys are also used to increase or decrease a selected value for a Direct Command (in the VECTOR4 menu) by 0.1 volts.
PgUp/PgDn	In a Direct Command (using VECTOR4 menu) and Set Gains, these keys will add (*PgUp*) or subtract (*PgDn*) one volt increments to or from a selected value.
Esc	ESC will either abort your selection or move cursor back to the previous choice or menu.
Space Bar	Selects an item identified by a square. As a result of pressing the *Space Bar* a cross sign X will appear on a selected square.
Home/End	Using the *Home/End* keys in a Direct Command (using VECTOR4 menu) increases (*Home*) or decreases (*End*) the direct output voltage by 0.01 volts.
Pause	Stops the operation of Signal Generator and freezes the last traced signals on the scope display.

Starting Mx4Pro

The preceding Fig. 3-2 depicts four selectable items on the Main Menu: Motor Technology, Power Technology, Sensor Technology and Dynamic Move. The first three selections are relevant only if you have the VECTOR4 drive control option.

Motor Technology	'Motor Technology' allows the user to choose a motor technology and set related parameters. (For VECTOR4 use only.)

Power Technology	'Power Technology' allows the VECTOR4-switching power stage interface parameters to be defined. (For VECTOR4 use only.)
Sensor Technology	'Sensor Technology' is used to define some encoder and motor parameters for VECTOR4. (For VECTOR4 use only.)
Dynamic Move	'Dynamic Move' contains all motion control functions and features of the Mx4 cnC++ card.

Although this is a manual for the Mx4 cnC++ controller, we have included a brief description of the first three main menu choices. If you have purchased Mx4 cnC++ only and are not interested in the VECTOR4 aspects of *Mx4Pro*, you may skip *Overview Of Mx4Pro Mx4 cnC++ Support* and go on to the *Running Mx4 cnC++ With Mx4Pro*.

Overview of Mx4Pro VECTOR4 Support

Mx4 cnC++ with the VECTOR4 option controls any combination of brushless DC, AC induction and brush-type DC motors. In addition to Mx4 cnC++'s capabilities, VECTOR4 performs all control functions commonly performed by DC or AC servo amplifier control boards. VECTOR4 customizes the control to motor, sensor, and power technologies. These three subjects are the first three selections of the *Mx4Pro* Main Menu.

Note: Performing a Mx4 cnC++-VECTOR4 command from one of these menu options will have no effect on a Mx4-only system.

Motor Technology

If you select **Motor Technology** from the Main Menu, you will see a menu asking for the axis to be specified. Use the *Arrow* keys to select the axis number and press *Enter*. On the right hand side of the monitor, a picture of three motors (brushtype DC on the top, AC induction motor in the middle, and brushless DC on the bottom) will appear (Fig. 3-3).

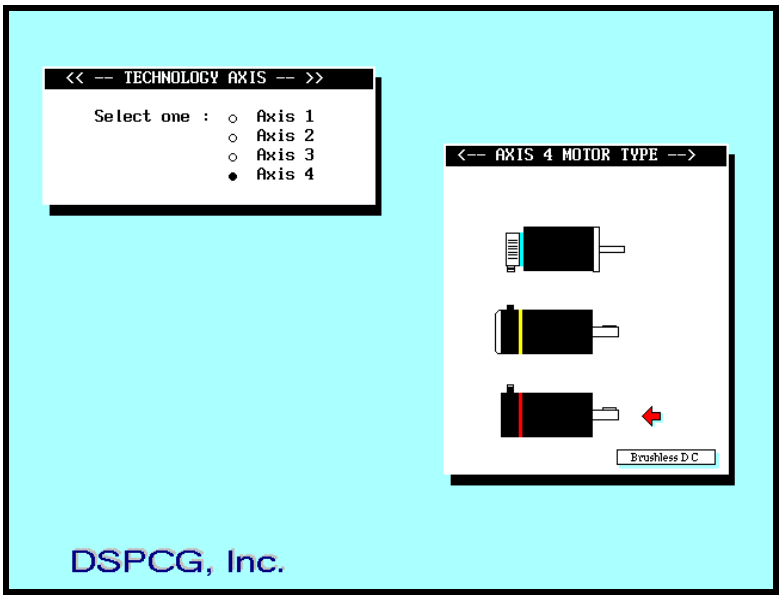


Fig. 3-3: Motor Technology Selection Screen (*For use with VECTOR4 only.*)

You must use *Arrow* keys to choose one of the three motor technologies and press *Enter*. If you are using an Mx4 cnC++-only system, this selection will have no effect. As a result of selecting **AC Induction** or **Brushless DC** motor technologies, a menu containing drive parameters will appear. Selecting an item on this menu will drop a window allowing you to enter the value(s) for a selected item(s). For example, you are allowed to enter PID gains if you select **Torque Gains** on this menu (see Fig. 3-4). To transmit your parameters to VECTOR4 you must select **Done** and press *Enter*. Pressing *Enter* on **Done** will take you back to the Main Menu.

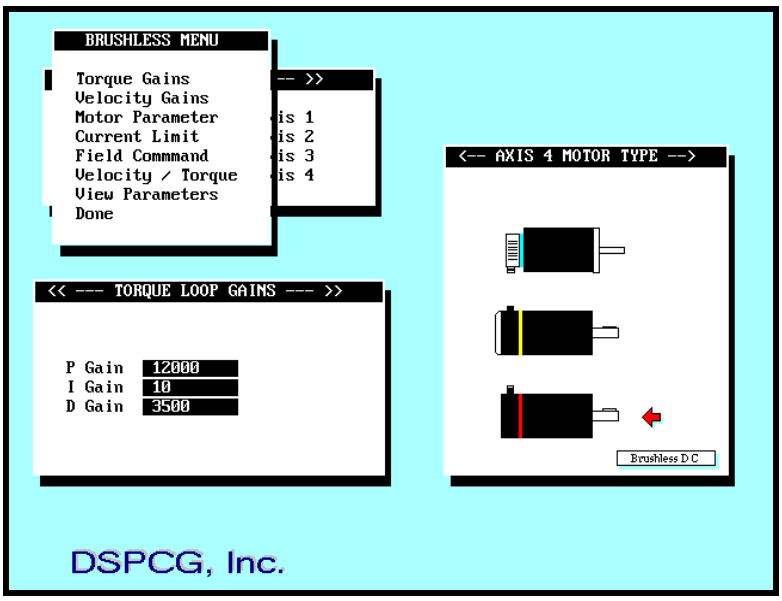


Fig. 3-4: Setting Torque Loop Gains for a Brushless DC Motor (*For use with VECTOR4 only*.)

Power Technology

Choosing **Power Technology** from the Main Menu allows you to enter the PWM frequency. A generic switching power stage merely follows VECTOR4's PWM command. Here again, to transmit your selected parameters you must select **Done** and press *Enter*.

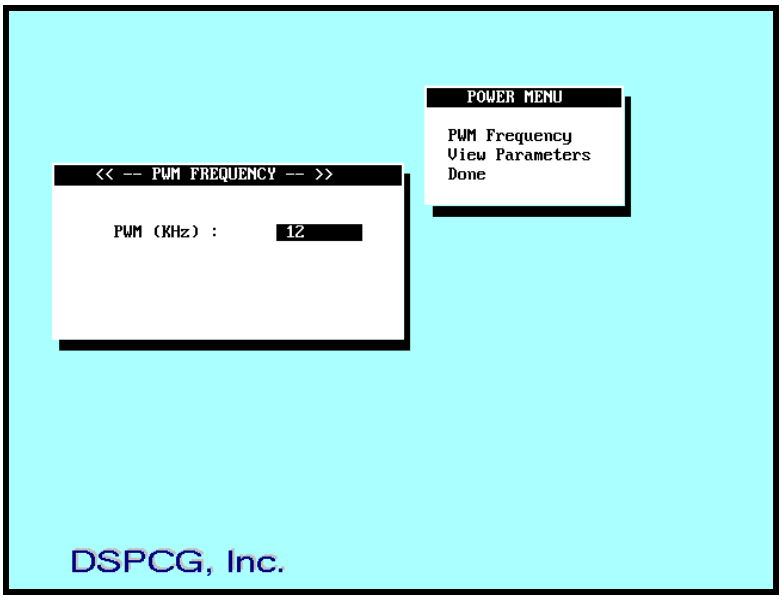


Fig. 3-4: Power Electronics Technology, PWM Frequency Selection (*For use with VECTOR4 only*.)

Sensor Technology

Sensor Technology enables the user to characterize the motor and encoder for VECTOR4. The first item is an entry for the number of encoder pulses per one turn of a shaft in a rotary application. Next is the number of motor poles. Industrial AC motors may have any number of poles from 2 to 20. The last item will inform the control of the mounting angle of the commutation sensors (please see the *Mx4 cnC++ User's Guide, VECTOR4*). To transmit parameters to VECTOR4, select **Done** and press *Enter*. Remember that if the selected technology is brushtype DC, there is no need for parameter entry in this part. This is due to the fact that VECTOR4 uses this information in an AC motor commutation.

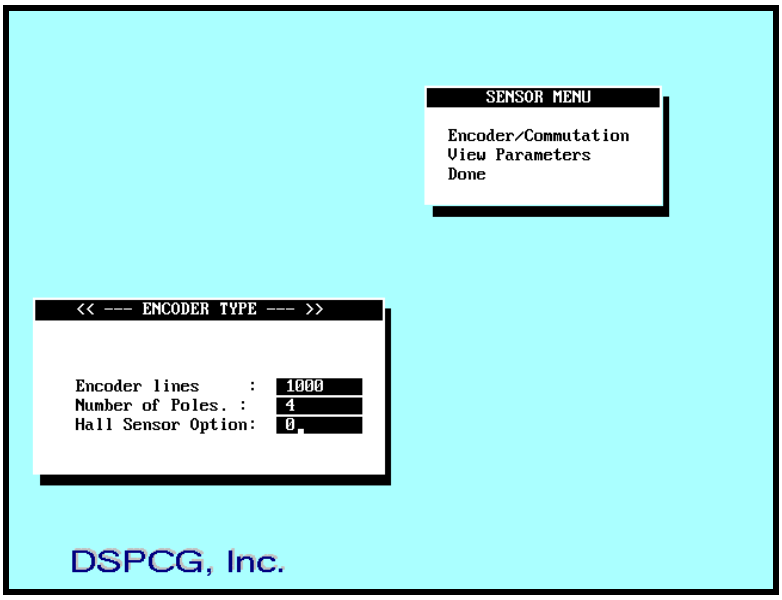


Fig. 3-5: Sensor Technology Screen, Setting Encoder Parameters (*For use with VECTOR4 only*.)

Summary of VECTOR4 Support

The first three Main Menu selections (**Motor Technology**, **Power Technology**, **Sensor Technology**) are used to program VECTOR4-option parameters. These parameters are typically programmed only once during the initialization of a Mx4 system. Again, performing a command from one of these menu options will have no effect on a Mx4-only system. Once these 'VECTOR4' parameters are set, the operation of a Mx4 system is equivalent to that of a Mx4-only system. That is, VECTOR4 is a 'transparent' interface between the Mx4 card and a generic switching power stage, driving AC or DC motors. The **Dynamic Move** selection allows us now to begin experimenting with motion control programming.

As a result of choosing **Dynamic Move** from the Main Menu, a window similar to Fig. 3-6 will appear on your monitor. At the top of this display you observe a text field with three columns and four rows. The three columns illustrate values for position, position error and velocity. The position and position error are described in "encoder edge counts" (with a quadrature encoder, one encoder pulse generates four encoder edges). The unit for velocity is "encoder edge counts per 200 µsec". Each row represents these parameters for one axis.

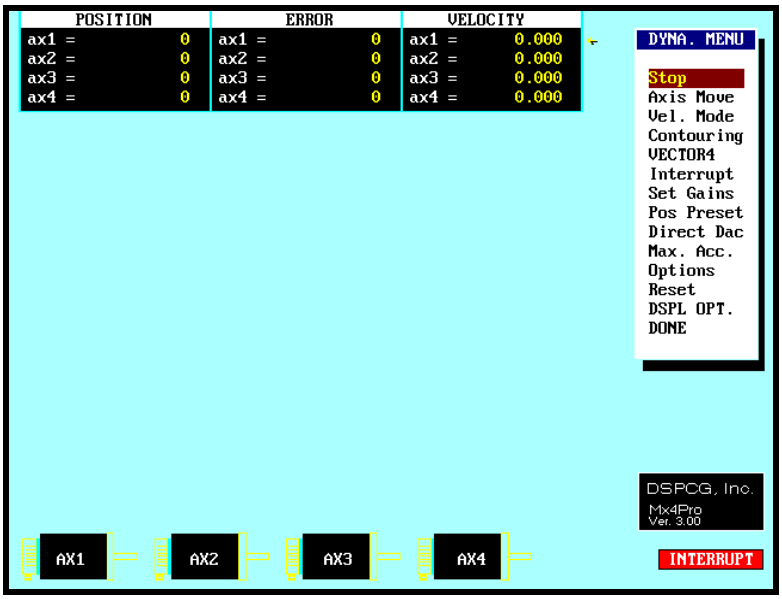


Fig. 5-1: Dynamic Move Main Menu Screen

On the right hand side of the screen you find the Dynamic Move Menu including these options:

```
Stop
Axis Move
Vel. Mode
Contouring
VECTOR4
Interrupt
Set Gains
Pos Preset
Direct DAC
Max. Acc.
Options
Reset
DSPL OPT.
DONE
```

The *Mx4Pro: Mx4 cnC++ Tuning Expert* manual will walk you through this menu showing various features of Mx4 cnC++.

This page intentionally left blank.

4 Programming The Mx4 cnC++

DSP Control Group has incorporated years of experience in the motion control industry developing Mx4 cnC++'s programming platform.

Host-Based Programming

Low-level Host-based programming entails real-time communication between the host computer and the Mx4 cnC++ card across the host computer bus. The host computer may read and write to the Mx4 cnC++ card as it would any computer peripheral. The user may choose the programming language of the host computer program. This host program includes the facilities to transfer commands to the Mx4 cnC++ card through the host bus, any conditional program code execution routines, PLC emulating code, an optional interrupt service routine to handle any enabled Mx4 cnC++ interrupts, Mx4 cnC++ system parameter readback routines and any other software features required for the application. With Host-based programming, an executable host program runs the operation of the Mx4 cnC++ card in real-time.

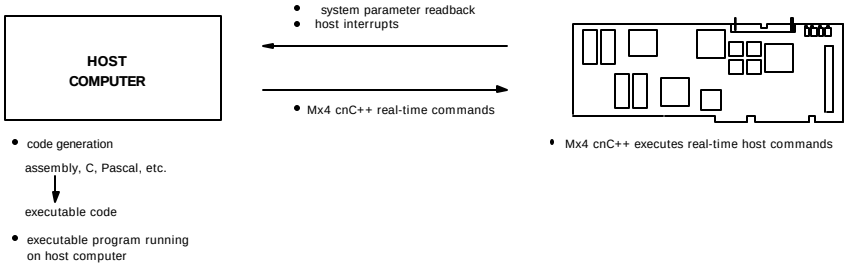


Fig. 4-1: Mx4 cnC++ Host-Based Programming

The Mx4 cnC++ Host-based programming platform includes two types of host-based commands:

Real Time Commands (RTCs)
Contouring Commands

Any combination of the two types of commands is possible for the four axes of control.

Real-Time Commands

Real Time Commands (RTCs) are transferred to Mx4 cnC++ through a "window" in the Mx4 cnC++ Dual Port RAM (DPR). Mx4 cnC++ polls the DPR for RTCs. An RTC is acted upon as soon as Mx4 cnC++ reads it. Multi-axis commands are executed simultaneously (not multiplexed), resulting in perfect sychronicity for multi-axis control. As soon as a new command is detected, Mx4 cnC++ executes it, possibly altering the effects of any previous commands that were not yet completed. The Mx4 cnC++ Host-based programming command set consists entirely of RTCs.

Contouring

Mx4 cnC++ supports two types of contouring: 2nd order contouring and cubic spline contouring. Contouring commands consist of segment move commands from which Mx4 cnC++ performs 2nd order or cubic spline interpolation. Contouring 'data' is transferred from the host to Mx4 cnC++ via a ring buffer in the DPR. See Fig. 4-1. Each segment move consists of a 32 bit position value and 32 bit velocity value for each axis included in the contouring motion. Mx4 cnC++ interpolates between the [position,velocity] points with programmable intervals. The 'commands' are executed in sequence, with execution commencing only when the previously commanded segment move is complete. A more detailed discussion of contouring commands can be found in Chapter 6 *Mx4 cnC++ Host-Based Programming*.

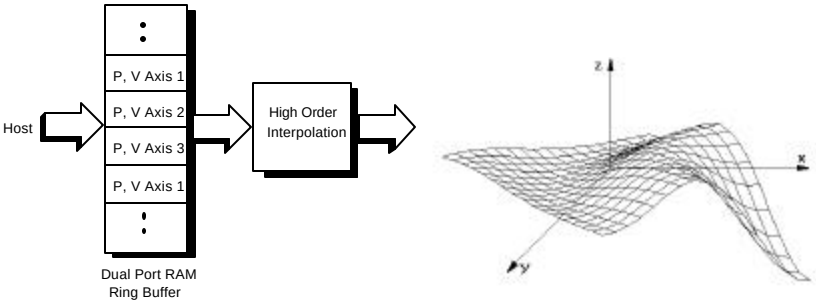


Fig. 4-3: Mx4 cnC++ Contouring with Three Axes

This page intentionally blank.

5 Mx4 cnC++ Host-Based Instruction Set

Host-Based Programming Command Set

The Mx4 cnC++ Host programming platform includes the following Real Time Commands (RTCs). These commands along with the previously mentioned contouring commands, yield a powerful and very flexible motion control programming platform. The Mx4 cnC++ RTCs are categorized as follows:

Initialization

Commands used to set-up and define system state variables and data reporting schemes are referred to as initialization commands.

COMMAND	DESCRIPTION
ABORTACC	specify abort maximum acceleration
HOME	preset position counters
HOMESFT	position counter reference shift
MAXACC	specify maximum acceleration
MTURN	define multi-turn position reporting
SYNC	configure Mx4 cnC++ as master or slave

Interrupt Control

Mx4 cnC++'s command set includes interrupt control instructions that allow interrupt conditions to be programmed and the ability to enable and disable Mx4 cnC++- host interrupts.

COMMAND	DESCRIPTION
BBINT	buffer breakpoint interrupt
DISABL	disable the interrupts #1
DISABL2	disable the interrupts #2
ENCOLOS	encoder fault / loss interrupt
FERHLT	following error / halt interrupt
FERINT	following error interrupt
INXINT	index pulse interrupt
MCENBL	motion complete interrupt
POSBRK	position breakpoint interrupt
POSFEED	positive feedback interrupt
PRBINT	general purpose probe interrupt

Trajectory Control

Trajectory control commands are those that specify closed-loop motion control.

COMMAND	DESCRIPTION
AXMOVE	axis move
STOP	stops the motion

System Diagnostic

System diagnostic commands allow the host to examine internal Mx4 cnC++ parameters and also provide debug support.

COMMAND	DESCRIPTION
PARREAD	parameter readback

Control Parameter

Instructions used to set state variable control parameters and to tune the control loops are classified as control parameter commands.

COMMAND	DESCRIPTION
CTRL	control law
KILIMIT	integral gain limit
OFFSET	amplifier offset cancellation
OUTGAIN	position loop output gain

Open Position Loop

Open position loop commands are motion commands based on velocity control or direct output control.

COMMAND	DESCRIPTION
DDAC	direct DAC command
VELMODE	velocity mode

Contouring

Contouring instructions are those related to the contouring mode of motion. These commands are used to define contouring parameters such as the contouring block transfer rate.

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

Filtering (optional)

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

I/O

COMMAND	DESCRIPTION
DISABORT	disable input abort processing
ENABORT	enable input abort processing
INPSTATE	configure logic state of inputs
OUTREL	output relay state

Reset

COMMAND	DESCRIPTION
RESET	reset Mx4 cnC++ controller card

Mx4 cnC++ RTC Instruction Set

The 38 Real Time Commands are listed in alphabetical order. Some of the description presented is technical information pertaining to the programming of the RTCs.

Note: Many instructions include the argument n ("a single byte bit coding the axes involved"). The format of n is:

n = (0000 axis 4 axis 3 axis 2 axis 1) B, where set bit(s) 3, 2, 1 or 0 specifies 4, 3, 2 or 1 respectively.

Mx4 cnC++ State Variables

Before programming the Mx4 cnC++ controller, knowledge of Mx4 cnC++'s state variables is necessary. The motion state variables are described below.

Acceleration Specified in encoder edge counts/(200 μs)^2. It is presented by a 16-bit unsigned number with 1 bit integer and 15 bits fraction.

i.e., acc = 077Bh = 0.0584 counts/(200 μs)^2

Following Error Specified in encoder edge counts and is presented by a 32-bit two's complement number with all 32 bits as integer.

Position Specified in encoder edge counts and is presented by a 32-bit two's complement number with all 32 bits as integer.

Velocity Specified in encoder edge counts/200 μs and is presented by a 27-bit two's complement number, sign extended to 32 bits. This value is partitioned as 16 bits integer and 16 bits fraction.

i.e., vel = 000A8000h = 10.50 counts/200 μs

Mx4 cnC++ Host-Based Programming Command Listing

The Mx4 cnC++ Host-based programming RTCs are listed in alphabetical order. Each command listing follows this format:

FUNCTION	indicates the command function
SYNTAX	order in which the command arguments must be written to the DPR Real Time Command buffer[1]
RTC CODE	real time command code
ARGUMENTS	command arguments, if any, are defined[2]
DESCRIPTION	explanation of command operation and functionality
SEE ALSO	listing of related commands
APPLICATION	some helpful suggestions describing which applications benefit from the command
EXAMPLE	an example illustrating the command in use

Note 1: See Chapter 6, *Mx4 cnC++ Host Programming ... RTCs & Contouring* for a detailed description of how RTCs are transmitted to the Mx4 cnC++ controller.

Note 2: Many commands include the argument n ("a single byte, bit coding the axes involved"). The bit coding is as follows:

n	bit 0	axis 1
	bit 1	axis 2
	bit 2	axis 3
	bit 3	axis 4
	bit 4-7	unused

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

ABORTACC

FUNCTION Abort Maximum Acceleration

SYNTAX ABORTACC(n, acc_1, ... , acc_4)

RTC CODE 86h

ARGUMENTS

n	a single byte, bit coding the axes involved.
acc_x	16 bit unsigned value specifying the maximum halting acceleration (deceleration) for axis x

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

DESCRIPTION

This command specifies the maximum halting acceleration (deceleration) for the axes specified. The maximum acceleration values are used in the following cases: FERHLT interrupt, ESTOP, probe interrupt and input abort processing.

Note: ABORTACC command will be ignored if the specified argument is zero.

SEE ALSO FERHLT, PRBINT, STOP, VELMODE

APPLICATION

This command sets the maximum possible deceleration for a mechanical actuator. This RTC is to set the deceleration rate for an emergency case. In contrast to the Mx4 RTC, ABORTACC 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 Mx4 for their acceleration/deceleration.

ABORTACC cont.

Command Sequence Example

```
ABORTACC ( )    ;set the abort maximum acceleration
CTRL ( )        ;make sure the system is in closed loop
FERHLT ( )      ;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 maximum acceleration for axes 2 and 3 of 0.5 encoder counts/200μsec^2.

(0.5) x 215 = 4000h

The values of the RTC argument are:

n : 06h
acc_2 : 4000h
acc_3 : 4000h

AXMOVE

FUNCTION Axis Move with Trapezoidal Trajectory

SYNTAX AXMOVE(n, acc_1, pos_1, vel_1, ... , acc_4, pos_4, vel_4)

RTC CODE 60h

ARGUMENTS

n	a single byte, bit coding the axes involved.
acc_x	16 bit acceleration for axis x
pos_x	32 bit end position for axis x
vel_x	32 bit slew rate for axis x

Note: Position and velocity are always presented in 2's complement format, but acceleration is an unsigned value.

Note: Velocity must be presented as a 27 bit 2's complement value which is sign extended to 32 bits. For example, the maximum positive velocity is 03FFFFFh and the maximum negative velocity is FC000000h.

Note: Velocity is partitioned into 16 bits integer and 16 bits fraction. Position is a 32 bit integer value, and acceleration is presented as 1 bit integer, 15 bits fraction.

DESCRIPTION

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

SEE ALSO 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. to make sure system can be
                ;stopped
CTRL ( )        ;set the gain values
KILIMIT ( )
AXMOVE ( )      ;run the system in axis move (linear trapezoidal) ;mode
.
.
MCENBL ( )      ;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 234567h and axis 2 to the target position of 112233h. Let's also assume that we want this move to be accomplished with the slew rate velocity of 200000h (200000h/2^{16} counts/200µsec) and acceleration of 150h (150h/2^{15} counts/(200µsec)^2) for both axes.

AXMOVE cont.

The values of the RTC arguments are:

n	:	03h
acc_1	:	0150h
pos_1	:	00234567h
vel_1	:	00200000h
acc_2	:	0150h
pos_2	:	00112233h
vel_2	:	00200000h

EXAMPLE 2

Assuming a current position of 0 for axis 4, we want to move axis 4 to the (negative) target position of FFAA0000h with a slew rate of FFE00000h (FFE00000h/2^{16} counts/200µsec)(negative velocity) and acceleration of 150h (150h/2^{15} counts/(200µsec)^2).

The values of the RTC arguments are:

n	:	08h
acc_4	:	0150h
pos_4	:	FFAA0000h
vel_4	:	FFE00000h

EXAMPLE 3

The host can issue a new axis move command before the previous one is completed. For example, assume the AXMOVE RTC of Example 1 is issued by the host. Now, the host changes its mind and decides to stop axis 2 at a new target position of 334455h with a new slew rate of 100000h. 100000h/2^{16} counts/200µsec) and a new acceleration of 200h (200h/2^{15} counts/(200µsec)^2). While the AXMOVE of Example 1 is in progress, the host issues the new command.

AXMOVE cont.

The values of the RTC arguments are:

n	:	02h
acc_2	:	0200h
pos_2	:	00334455h
vel_2	:	00100000h

BBINT

FUNCTION Buffer Breakpoint Interrupt

SYNTAX BBINT(buffbrk)

RTC CODE 61h

ARGUMENTS

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

DESCRIPTION

This command will cause an interrupt when the number of instructions in the ring buffer falls below a preset breakpoint. The buffer breakpoint interrupt status will appear in bit 0 of the DPR interrupt flag location 03FEh, 7FEh. This bit gets set if the buffer breakpoint interrupt occurs.

SEE ALSO DISABL

APPLICATION

This command must be used in contouring applications. To maintain continuity in a contouring application, Mx4 must be constantly updated by the host processor a set of new (position/velocity) points on a contour. Since no application can afford to run out of points, the host must set the BBINT 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.

BBINT cont.

Command Sequence Example

```
MAXACC ( )      ;make sure that a system can be stopped
CTRL ( )        ;set the gains
KILIMIT ( )
.               ;load the ring buffer with contouring points,
.               ;(position and speed)
BTRATE ( )      ;set the block transfer rate to 5, 10, 15 or 20ms
BBINT ( )       ;set the breakpoint in buffer
.
.
START (n)       ;start contouring
```

EXAMPLE

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

The value of the RTC argument is:

buffbrk : 1Eh

BTRATE

FUNCTION Set 2nd Order Contour Block Transfer Rate

SYNTAX BTRATE(m)

RTC CODE 73h

ARGUMENTS

m a byte which 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 5ms per point
m=1	block transfer rate is 10ms per point
m=2	block transfer rate is 15ms per point
m=3	block transfer rate is 20ms 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 10ms, the time interval between each point in the ring buffer is '10ms' (e.g. the DSP will interpolate each point for 10ms).

Note: Host should not adjust the block transfer rate when contouring is in process.

Note: The default block transfer rate is set at 5ms 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 20ms), Mx4 will internally perform a high-order interpolation of the points breaking them down to 200 μsec.

Command Sequence Example

See BBINT

EXAMPLE

Set a contouring interpolation interval of 10msec.

The value of the RTC argument is:

m : 01h

CTRL

FUNCTION Control Law Parameters

SYNTAX CTRL(n, par_{11}, ... , par_{14}, ... , par_{n1}, ... , par_{n4})

RTC CODE 62h

ARGUMENTS

n	a single byte, bit coding the axes involved.
$par_{x1}(K_i)$	16 bit unsigned value for the first control parameter for axis x.
$par_{x2}(K_p)$	16 bit unsigned value for the second control parameter for axis x.
$par_{x3}(K_f)$	16 bit unsigned value for the third control parameter for axis x.
$par_{x4}(K_d)$	16 bit unsigned value for the fourth control parameter for axis x.

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 rage 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; par1, par2, par3, and par4 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
K_f	2^{15}
K_p	2^7
K_i	1
K_d	(17×2^{11})
Output Loop Gain	20 (volts) / 2^{17}

For example,

50 counts of position error and K_p of 1 (other gains are zero) will result in an output voltage of 976 millivolts.

i.e. $50 \times 1 \times 2^7 \times 20 / (2^{17}) = 0.976$

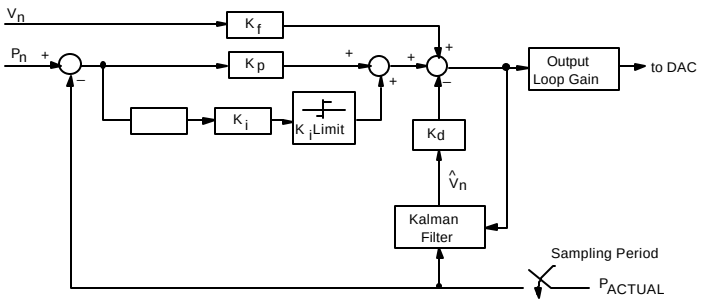


Figure 5-2: Block Diagram of Control Law

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 Chapter 3 on *Mx4 cnC++Pro*. *Running MX4 with Mx4 cnC++Pro* of that chapter will help you understand the significance of gains in tuning. Please read this section even if you cannot run *Mx4 cnC++Pro* 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

The values of the RTC arguments are:

n : 0Ah
par_{21} : 0064h
par_{22} : 0FA0h
par_{23} : 0BB8h
par_{24} : 09C4h
par_{41} : 0014h
par_{42} : 1F40h
par_{43} : 157Ch
par_{44} : 1658h

CUBIC_RATE

FUNCTION Set Cubic Spline Point Transfer Rate

SYNTAX CUBIC_RATE (m)

RTC CODE 89h

ARGUMENTS

m — a 16-bit 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.

DESCRIPTION

This command sets the point transfer rate for the cubic spline. The "transfer rate" sets the interval between two adjacent points in the cubic spline 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 ms 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 RTC 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 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	$5 \leq m < 84$ points
for b.t. rate of 5 ms	$1 \leq m < 84$ points

SEE ALSO BBINT, BTRATE, CUBIC_SCALE

APPLICATION

Refer to *Cubic Spline Application Notes*.

EXAMPLE

Using cubic spline interpolation create 16, 32, 64 and 128-point circles.

The following shows the position and velocity values for 16 uniformly spaced points on a circle.

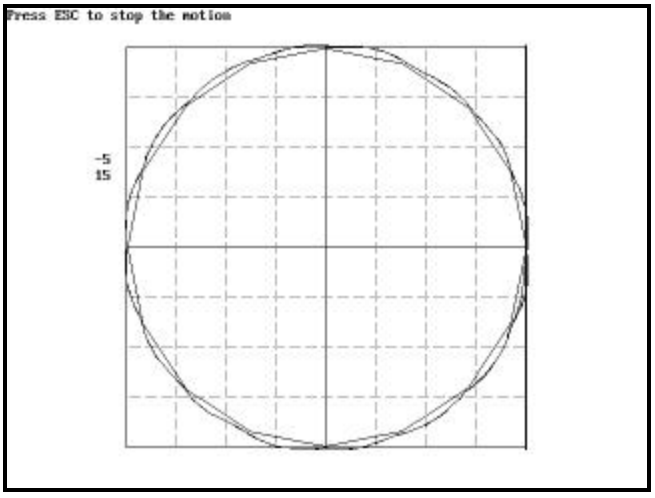
16-point Circle

Point	pos_x	actual_vel_x	coded_vel_x
x1	2500 (0x000009C4h)	0 (0x00000000h)	0x30000000h
x2	2310 (0x00000906h)	-61554 (0xFFFF0F8Eh)	0x3FFF0F8Eh
:	:	:	:
x16	2310 (0x00000906h)	+61554 (0x0000F072h)	0x3000F072h

CUBIC_RATE cont.

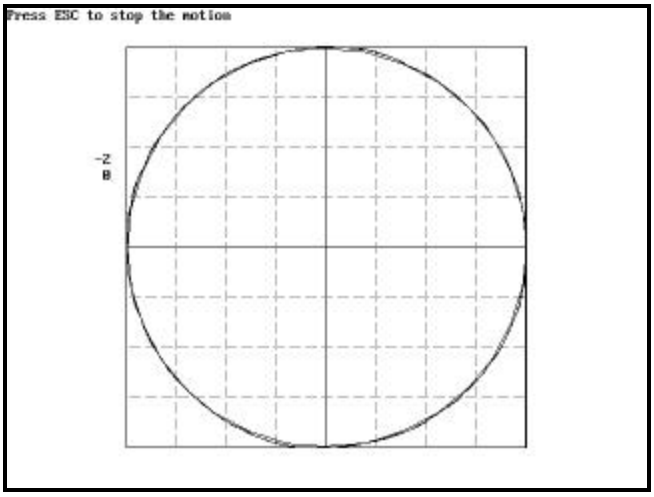
Point	pos_y	actual_vel_y	coded_vel_y
x1	0 (0x00000000h)	160850 (0x00027452h)	0x30027452h
x2	957 (0x000003BDh)	148610 (0x00024482h)	0x30024482h
:	:	:	:
x16	-957 (0xFFFFFC43h)	148610 (0x00024482h)	0x30024482h

To generate a circle, these points must be written to Mx4's cubic spline ring buffer and CUBIC_RATE must be executed. The CUBIC_RATE argument determines the interval between two points of the ring buffer. If the number of points on a trajectory (i.e., circle) exceeds the size of the ring buffer, the BBINT (buffer breakpoint interrupt) RTC must be used. This command, sets the breakpoint where the host must load more points to the ring buffer. This way the CPU will refresh the ring buffer on a continuous basis. For comparison, the following figures illustrate the circles created by 16, 32, 64 and 128 points in a cubic spline interpolation. It takes 1.28 seconds to complete these circles.

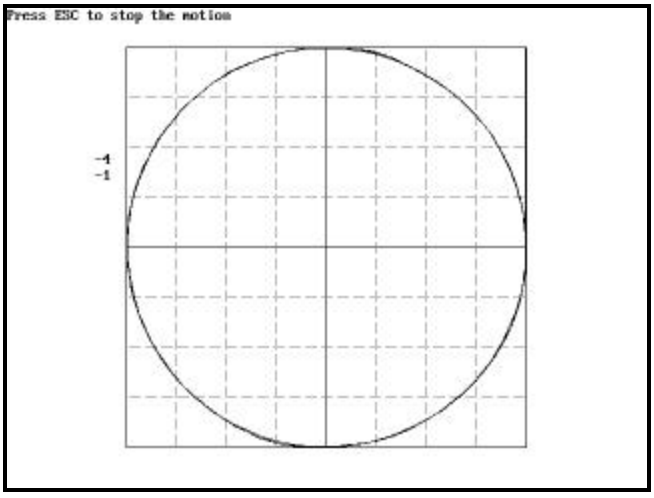


16 points; b.t. rate = 80 ms

CUBIC_RATE cont.

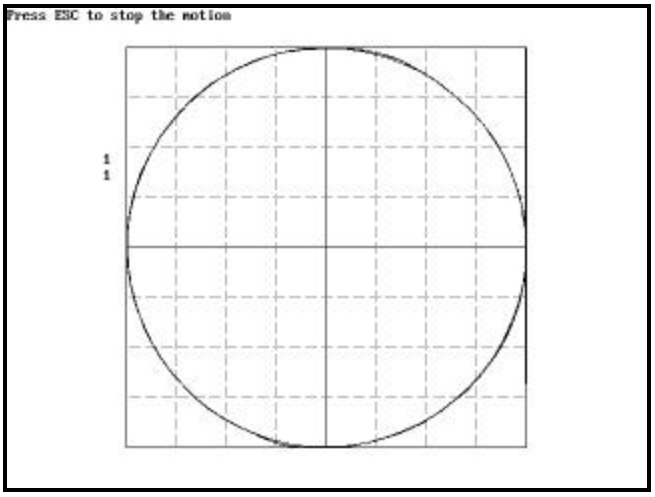


32 points; b.t. rate = 40 ms



64 points; b.t. rate = 20 ms

CUBIC_RATE cont.



128 points; b.t. rate = 10 ms

CUBIC_SCALE

FUNCTION Scale Cubic Spline Data Points

SYNTAX CUBIC_SCALE (n, pv_mult_1, pos_shift_1, ... , pv_mult_4, pos_shift_4)

RTC CODE 8Bh

ARGUMENTS

n — a single byte, bit coding the axes involved

pv_multx — position / velocity scaling multiplier for axis x. This is a 16-bit two's complement number with one sign bit, one integer bit, and fourteen bits fraction.

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

DESCRIPTION

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

SEE ALSO CUBIC_RATE

APPLICATION See *Cubic Spline Application Notes*

EXAMPLE

Set a scale of 0.5 for all axis 2 cubic spline data points. No position shift is desired.

The values of the RTC arguments are:

n : 0x02h
pv_mult_2 : 0x2000h
pos_shift_2 : 0x00000000h

DDAC

FUNCTION Direct DAC Output

SYNTAX DDAC(n, val_1, ... , val_4**)**

RTC CODE 63h

ARGUMENTS

n — a single byte, bit coding the axes involved.

val_x — 16 bit value specifying the 16 bit DAC output voltage for axis x.

The values range as follows:

FFFFh	:	-10(1/32768)v output
:		
8000h	:	-10v output
7FFFh	:	+10v output
:		
0000h	:	0v output

DESCRIPTION

Specifies a bipolar analog signal ranging from -10 to +10 volts with a resolution of 0.3 millivolts.

SEE ALSO none

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).

DDAC cont.

Command Sequence Example

No preparation is required before running this instruction.

EXAMPLE

Output +3.7 volts to the axis 4 DAC (DAC4 MX4 connector signal).

$$(\tfrac{+3.7}{10}) \times 7FFFh = 2F5Ch$$

The values of the RTC arguments are:

n	:	08h
val_4	:	2F5Ch

DISABL

FUNCTION Disable Interrupts

SYNTAX DISABL(n, m_1, ... , m_4)

RTC CODE 64h

ARGUMENTS

n — a single byte, bit coding the axes involved.

m_x — a single byte, bit mapping the interrupts to disable for axis x (setting a bit to one indicates disabling an interrupt).

bit 7 : -
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 BBINT, DISABL2, PRBINT, FERHLT, FERINT, INXINT, MCENBL, 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.

DISABL cont.

Command Sequence Example

No preparation is required before running this instruction.

EXAMPLE

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

The values of the RTC arguments are:

n	:	05h
m_1	:	04h
m_3	:	20h

DISABL2

FUNCTION Disable Interrupts

SYNTAX DISABL2(n, m_1, ... , m_4)

RTC CODE 5Ah

ARGUMENTS

n — a single byte, bit mapping the axes involved.

m_x — a single byte, bit mapping the interrupts to disable axis x (setting a bit to one indicates disabling an interrupt).

bit 7 : not used
: : not used
bit 2 : not used
bit 1 : encoder loss
bit 0 : positive feedback

DESCRIPTION

This command disables some of the servo control card interrupts.

SEE ALSO DISABL, ENCOLOS, POSFEED

APPLICATION

In servo applications checking for failures such as encoder loss or positive feedback loop is a task performed on power-up. Once an application is assured of proper feedback polarity, the encoder loss and positive feedback interrupts may be disabled throughout the entire application.

DISABL2 cont.

Command Sequence Example

No preparation is required before running this instruction.

EXAMPLE

Disable the previously enabled axis encoder loss and axes 1 and 4 positive feedback interrupts.

n	:	09h
m_1	:	03h
m_4	:	01h

DISABORT

FUNCTION Disable Input Abort Processing

SYNTAX DISABORT(n, $dabort_1$, ... , $dabort_4$)

RTC CODE 57h

ARGUMENTS

n a single byte, bit coding the axes involved.
$dabort_x$ a single byte, bit mapping the input disables for axis x.

bit=0 : no change in enable/disable status
bit=1 : disable the interrupt processing

bit 7 : unused
bit 6 : unused
bit 5 : gen. purpose input 4, gen. purpose input 5 input*
bit 4 : unused
bit 3 : unused
bit 2 : unused
bit 1 : - O.T. input for axis x
bit 0 : + O.T. input for axis x

Note: * Bit 5 is used to select the enable/disable status of inputs gen. purpose input and gen. purpose input 5. When axis 1 is selected, bit 5 corresponds to gen. purpose input 4. When axis 2 is selected, bit 5 corresponds to gen. purpose input 5. If either axis 3 or axis 4 is selected, bit 5 of the corresponding bytes is an "unused" bit.

DESCRIPTION

This command allows the user to disable the interrupt and interrupt processing for the specified inputs.

SEE ALSO DISABL, DISABL2, ENABORT

DISABORT cont.

APPLICATION

This RTC is used when one or several inputs of MX4 CNC++000MB-IO need to be used as general purpose inputs. Using this command disables an interrupt (as well as interrupt processing) that occurs when an input signal is set.

Command Sequence Example

No preparation is required before running this instruction.

EXAMPLE

Disable inputs + O.T. and - O.T. for axis 1 and 2:

The value of the RTC argument is:

n	:	03h
$dabort_1$	:	03h
$dabort_2$	:	03h

ENABORT

FUNCTION Enable Input Abort Processing

SYNTAX ENABORT(n, m_1, $eabort_1$, ... , m_4, $eabort_4$)

RTC CODE 58h

ARGUMENTS

n — a single byte, bit coding the axes involved.

m_x — a single byte, bit coding the axes to be halted upon receipt of an active input condition on an enabled input of axis x.

$eabort_x$ — a single byte, bit mapping the input enables for axis x.

bit=0 : no change in enable/disable status
bit=1 : enable the interrupt processing

bit 7 : unused
bit 6 : unused
bit 5 : gen. purpose input 4, gen. purpose input 5 input*
bit 4 : unused
bit 3 : unused
bit 2 : unused
bit 1 : - O.T. input for axis x
bit 0 : + O.T. input for axis x

Note: * Bit 5 is used to select the enable/disable status of inputs gen. purpose input 4 and gen. purpose input 5. When axis 1 is selected, bit 5 corresponds to gen. purpose input 4. When axis 2 is selected, bit 5 corresponds to gen. purpose input 5. If either axis 3 or axis 4 is selected, bit 5 of the corresponding bytes is an "unused" bit.

ENABORT cont.

DESCRIPTION

This command allows the user to enable the interrupt and interrupt processing for the specified inputs. If any enabled active input condition of axis x is received by MX4, the axes specified by the m_x argument will be halted (similar to ESTOP). The interrupt condition is recorded in DPR interrupt status register location 009h. The DPR status register location 00Dh will identify the axis or axes responsible. DPR locations 094h-096h identify the input status in real-time (yielding the interrupt type and source). Bit 6 of DPR locations 3FEh, 7FEh is also set.

SEE ALSO DISABORT, INPSTATE, PRBINT

APPLICATION

This command in conjunction with limit switches mounted on a machine may be used to bring a system (or part of a system) to an immediate stop. Enabling the abort aimed at a particular axis will bring that axis to a halt. This happens when input(s) selected by this command is (are) set.

Command Sequence Example

No preparation is required before running this instruction.

EXAMPLE

Enable abort for + O.T. input of axis 1 and - O.T. input of axis 2. When the axis 1 input is set, axes 3 and 4 are to be stopped. When the axis 2 input is set, all of the axes are to be stopped.

n	:	03h
m_1	:	0Ch
eabort$_1$	:	01h
m_2	:	0Fh
eabort$_2$	:	02h

ENCOLOS

FUNCTION Encoder Loss Interrupt

SYNTAX ENCOLOS(n)

RTC CODE 5Ch

ARGUMENTS

n a single byte, bit coding the axes involved.

DESCRIPTION

This command enables the encoder loss interrupt for the specified axes. Encoder loss interrupt is generated if the following conditions are met (for the axis in question):

1. Following Error is > 2000 counts
2. If the command position changes, the actual position does not change.
3. The above 3 conditions hold for 0.3 seconds

The DPR interrupt status locations 009h and 00Bh record the occurrence and source of this interrupt. Bit 6 of DPR locations 3FEh, 7FEh is also set.

SEE ALSO DISABL2

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 loss interrupt for both axis 3 and axis 4.
The value of the RTC argument is:

n : 0Ch

FERHLT

FUNCTION Following Error Interrupt and Halt

SYNTAX FERHLT(n, fer_1, ... , fer_4)

RTC CODE 66h

ARGUMENTS

n	a single byte, bit coding the axes involved.
fer_x	16 bit unsigned following error for axis x.

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 abort maximum acceleration rate. This interrupt condition is recorded in DPR interrupt status register location 000h. The DPR status register location 001h reveals the axis(s) responsible. Bit 1 of DPR locations 3FEh, 7FEh is also set.

Note: FERHLT command will be ignored if the respective axis abort maximum acceleration is zero.

Note: Following error / halt interrupt is not disabled after it occurs. The host is responsible for disabling the interrupt.

SEE ALSO DISABL, FERINT, ABORTACC

APPLICATION

Applications of this command are similar to FERINT. 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 ABORTACC. Please note that this command is not appropriate to prevent system run-away in case of encoder loss - since in the absence of encoder, the system cannot be stopped reliably.

FERHLT cont.

Command Sequence Example

```
ABORTACC ( )    ;make sure system can be stopped
CTRL ( ) ;these instructions enable system to stop motion
KILIMIT ( )         ;set gains
.
.
FERHLT ( )
```

EXAMPLE

Enable a following error/halt interrupt for axis 3 with a threshold of 100 encoder counts.

The values of the RTC arguments are:

n	:	04h
fer_3	:	0064h

FERINT

FUNCTION Following Error Interrupt

SYNTAX FERINT(n, fer_1, ... , fer_4)

RTC CODE 67h

ARGUMENTS

n	a single byte, bit coding the axes involved.
fer_x	16 bit unsigned following error for axis x.

DESCRIPTION

Upon the execution of this command, if at any time the following error for a specified axis exceeds it's 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(s) responsible. Bit 1 of DPR locations 3FEh, 7EFh is also set.

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

SEE ALSO DISABL, FERHLT

APPLICATION

This command may be used in all applications for two main reasons. First, FERINT reports a run-away or any other out-of-control condition. Second, it makes sure that position error is within a specified (a programmed argument for FERINT) tolerance.

FERINT cont.

Command Sequence Example

No preparation is required before running this instruction.

EXAMPLE

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

The values of the RTC arguments are:

n	:	01h
fer_1	:	00C8h

HOME

FUNCTION Preset Position Counter

SYNTAX HOME(n, $pset_1$, ... , $pset_4$)

RTC CODE 68h

ARGUMENTS

n	a single byte, bit coding the axes involved.
$pset_x$	32 bit two's complement value to preset the axis x position counter.

DESCRIPTION

This command will define the present position point for the axes specified.

Note: HOME command will automatically disable the position breakpoint interrupt (if enabled). HOME can be executed only when the axes specified are not in motion.

SEE ALSO HOMESFT, POSBRK

APPLICATION

This command is useful when the position counter must be forced to a new value. This command may be used in the establishment of a new reference potion. Please also see HOMESFT.

HOME cont.

Command Sequence Example

No preparation is required before running this instruction.

EXAMPLE

Set the present position of axis 4 to 50,000 counts.

The values of the RTC arguments are:

n	:	08h
$pset_4$	:	0000C350h

HOMESFT

FUNCTION Home Reference Shift

SYNTAX HOMESFT(n, $psft_1$, ... , $psft_4$)

RTC CODE 5Dh

ARGUMENTS

n	a single byte, bit coding the axes involved.
$psft_x$	32 bit two's complement value to add to the axis x position counter.

DESCRIPTION

This command will shift the present position point for the axes specified.

Note: HOMESFT command will automatically disable the position breakpoint interrupt (if enabled) of the specified axes.

SEE ALSO HOME, 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 INXINT and HOME.

HOMESFT cont.

Command Sequence Example

No preparation is required before running this instruction.

EXAMPLE

The current axis 1 position is 100h. Shift the axis 1 position to 20100h. The current axis 3 position is 1010h. Shift the axis 3 position to 1000h.

The values of the RTC arguments are:

n	:	05h
$psft_1$	:	00020000h
$psft_3$	:	FFFFFFF0h

INPSTATE

FUNCTION Configure Logic State of Inputs

SYNTAX INPSTATE(inp_1, inp_2, inp_3)

RTC CODE 88h

ARGUMENTS

inp_1 a single byte, coding the logic state of inputs.

bit=0	:	active LOW input
bit=1	:	active HIGH input

bit 7	:	axis 4	-O.T. input
bit 6	:	axis 3	-O.T. input
bit 5	:	axis 2	-O.T. input
bit 4	:	axis 1	-O.T. input
bit 3	:	axis 4	+O.T. input
bit 2	:	axis 3	+O.T. input
bit 1	:	axis 2	+O.T. input
bit 0	:	axis 1	+O.T. input

inp_2 a single byte, unused (i.e., set to 00h)

inp_3 a single byte, coding the logic state of inputs.

bit=0	:	active LOW input
bit=1	:	active HIGH input

bit 6-7	:	unused
bit 5	:	general purpose input 5
bit 4	:	general purpose input 4
bit 0-3	:	unused

INPSTATE cont.

DESCRIPTION

This command allows the user to define the logic state of the Mx4 cnC++ inputs. Each input may be configured as active LOW or active HIGH (TTL logic levels) (the Mx4 cnC++ inputs are level sensitive).

Note: At power-up and reset, Mx4 cnC++ inputs default as active LOW.

SEE ALSO ENABORT

EXAMPLE

Configure the +O.T. inputs of axes 1-4 as active HIGH inputs. The remaining inputs are to be configured as active LOW.

The value of the RTC arguments is:

inp_1 : 0Fh
inp_2 : 00h
inp_3 : 00h

INXINT

FUNCTION Index Pulse Interrupt

SYNTAX INXINT(n)

RTC CODE 69h

ARGUMENTS

n a single byte, bit coding the only axis involved.

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 results in the generation of an interrupt and registration of 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 locations 3FEh, 7EFh is also set.

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

Note: The index pulse interrupt and general purpose external interrupt CAN BE ENABLED simultaneously.

SEE ALSO DISABL, HOME, HOMESFT

APPLICATION

This command is used in homing applications. As a result of this instruction, Mx4 cnC+ 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 HOME to perform homing for linear table (or other index-based) position calibration.

INXINT cont.

Command Sequence Example

No preparation is required before running this instruction.

EXAMPLE

Enable the index pulse interrupt for axis 4.

The value of the RTC argument is:

n : 08h

KILIMIT

FUNCTION Integral Gain Limit

SYNTAX KILIMIT(n, val_1, ... , val_4)

RTC CODE 74h

ARGUMENTS

n — a single byte, bit coding the axes involved.

val_x — a single byte value specifying the limit of the integral action for each axis.

Note: 0 <= val <= 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 0 (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 gains
KILIMIT ( )        ;this instruction may be used before or after CTRL
```

EXAMPLE

Set a maximum limit on the integral action of axis 2.

The values of the RTC arguments are:

n	=	02h
val_2	=	0Eh

LOW_PASS (option)

FUNCTION Implement Low Pass Filter at Controller Output

SYNTAX LOW_PASS (n, $Freq_1$, ..., $Freq_4$)

RTC 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

$0 \leq freq_x \leq 1850$

DESCRIPTION

This command implements a low pass filter at the controller output for the specified axis.

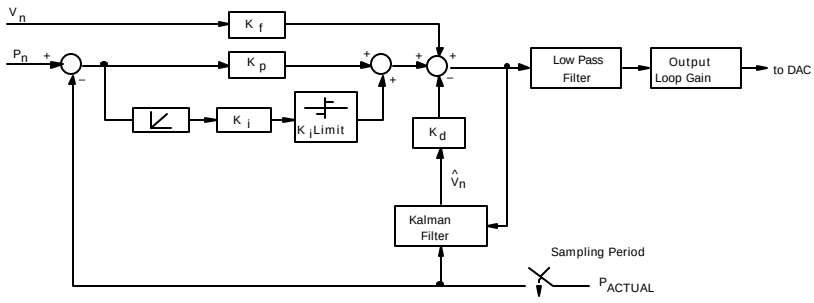


Fig. 4-2: Mx4 Block Diagram with Low Pass Filter

LOW_PASS cont.

The low pass filter implements the following transfer function:

$$G(s) = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n \cdot s + \omega_n^2}$$

where, $\omega_n = 2\pi f_n$, f_n = cut-off frequency, and $\zeta = 0.6$

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

LOW_PASS cont.

EXAMPLE: RTC Programming Low Pass

The LOW_PASS RTC uses the coded values for low pass frequency. Table 4-1 shows these coded values. Use of the index table is only necessary with RTCs.

Set a low pass filter at 275 Hz for axis 3.

The following shows the DPR's byte stream:

```
3c2   xxh    ;command code
3c3   xxh    ;axis 3
3c4   0Ah    ;index to element 10 of frequency table (275 Hz)
```

FREQ (Hz)	FREQ Index
disable filter	0
50	1
75	2
100	3
125	4
150	5
175	6
200	7
225	8
250	9
275	10
300	11
325	12
350	13
375	14
400	15
425	16
450	17
475	18
500	19
550	20
600	21
650	22
700	23

FREQ (Hz)	FREQ Index
750	24
800	25
850	26
900	27
950	28
1000	29
1050	30
1100	31
1150	32
1200	33
1250	34
1300	35
1350	36
1400	37
1450	38
1500	39
1550	40
1600	41
1650	42
1700	43
1750	44
1800	45
1850	46

Low Pass Filter Frequency Index

MAXACC

FUNCTION Maximum Acceleration

SYNTAX MAXACC(n, acc_1, ... , acc_4)

RTC CODE 71h

ARGUMENTS

n	a single byte, bit coding the axes involved.
acc_x	16 bit unsigned value specifying the maximum acceleration / deceleration for axis x.

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

DESCRIPTION

This command specifies the maximum acceleration / deceleration for the axes specified. The maximum acceleration values are used with the VELMODE and STOP RTCs.

Note: MAXACC command will be ignored if the specified argument is zero.

SEE ALSO STOP, VELMODE

APPLICATION

This command sets the maximum acceleration affordable by servo drive and motor combination. It is useful to program this parameter such that the system will not go to control saturation during the VELMODE or STOP command.

MAXACC cont.

Command Sequence Example

```
MAXACC ( )      ;make sure system can be stopped
CTRL ( )        ;set 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 / $(200\mu sec)^2$.

$$(0.25) \times 2^{15} = 2000h$$

The values of the RTC arguments are:

n	:	06h
acc_2	:	2000h
acc_3	:	2000h

MCENBL

FUNCTION Motion Complete Interrupt

SYNTAX MCENBL(n)

RTC CODE 65h

ARGUMENTS

n a single byte, bit coding the axes involved.

DESCRIPTION

This command enables the motion complete interrupt for the axes specified. The motion complete interrupt is generated when any motion 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 locations 3FEh, 7FEh is also set.

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

SEE ALSO DISABL

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.

MCENBL cont.

Command Sequence Example

See AXMOVE and STOP

EXAMPLE

Enable the motion complete interrupt for all four axes.

The value of the RTC argument is:

n : 0Fh

MTURN

FUNCTION Multi-Turn Position Reporting

SYNTAX MTURN(n, m_1, ... , m_4)

RTC CODE 82h

ARGUMENTS

n — a single byte, bit coding the axes involved.

m_x — a positive 16 bit value specifying the multi-turn base in encoder counts.

$0 \leq m_x \leq 32768$

DESCRIPTION

Multi-turn position reporting for each axis is available in DPR locations 097h - 0A6h (see *Parameter Updates, Dual Port RAM Partitioning*). This command allows the multi-turn base for specified axes to be programmed.

Multi-turn positions are calculated as offsets from position 0 described in terms of the number of turns and fraction of complete turn (described in terms of encoder counts) to reach the current actual position value. The multi-turn base is defined as the number of encoder counts per one 'multi-turn' turn.

For example, with a multi-turn base of 1000 encoder counts and an actual position of -32,555 counts, the multi-turn position values in the DPR will yield:

MTURN : -32
MFRAC : -555

SEE ALSO none

MTURN cont.

APPLICATION

This command will change the numerical base for the position of an axis to a programmable value. For example, in spindle applications, the number of turns (integer as well as fractional part) can be recorded. That is, position may be monitored as a function of the shaft's angular position.

Command Sequence Example

```
MAXACC ( )      ;make sure system can be stopped
CTRL ( )        ;set gains
KILIMIT ( )
.
.
AXMOVE ( )      ;run system in axis move (linear trapezoidal) mode
MTURN ( )
```

EXAMPLE

Set a multi-turn base of 1000 encoder counts for axis 2.

The values of the RTC arguments are:

n : 02h
m_2 : 03E8h

NOTCH (option)

FUNCTION Implement Notch Filter at Controller Output

SYNTAX NOTCH(n, $freq_1$, q_1, ..., $freq_4$, q_4)

RTC 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

$0 \leq freq_x \leq 1650$ Hz

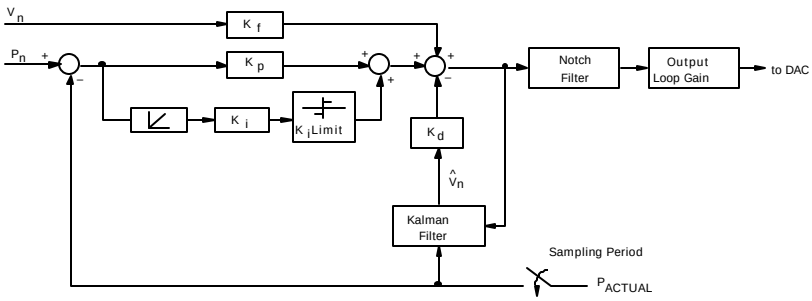
q_x — unsigned value specifying the notch filter quality factor for axis x

$q_x = 1$ ~25% bandwidth filter
$q_x = 2$ ~10% bandwidth filter

NOTCH cont.

DESCRIPTION

This command implements a notch filter at the controller output for the specified axis.



Mx4 Block Diagram with Notch Filter

The notch filter implements the transfer function:

$$G(s) = \frac{s^2 + \omega_n^2}{s^2 + \frac{\omega_n}{Q}s + \omega_n^2}$$

where, $\omega_n = 2\pi f_n$ 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

NOTCH cont.

EXAMPLE: RTC Programming Notch

The NOTCH RTC uses the coded values for both notch frequency and notch quality factor, q. Table 4-1 shows these coded values. Use of the index table is only necessary with RTCs.

Set a notch filter at 290 Hz with a wide bandwidth for axis 3.

The following shows the DPR's byte stream:

```
3c2   8Eh    ;command code
3c3   04h    ;axis 3
3c4   2Bh    ;index to element 43 of freq table (290 Hz)
3c5   0      ;index to wide bandwidth notch filter
```

NOTCH cont.

FREQ (Hz)	FREQ Index	FREQ (Hz)	FREQ Index
Disable notch	0	410	40
20	1	420	41
30	2	430	42
40	3	440	43
50	4	450	44
60	5	460	45
70	6	470	46
80	7	480	47
90	8	490	48
100	9	500	49
110	10	510	50
120	11	520	51
130	12	530	52
140	13	540	53
150	14	550	54
160	15	560	55
170	16	570	56
180	17	580	57
190	18	590	58
200	19	600	59
210	20	610	60
220	21	620	61
230	22	630	62
240	23	640	63
250	24	650	64
260	25	660	65
270	26	670	66
280	27	680	67
290	28	690	68
300	29	700	69
310	30	710	70
320	31	720	71
330	32	730	72
340	33	740	73
350	34	750	74
360	35	760	75
370	36	770	76
380	37	780	77
390	38	790	78
400	39	800	79

Notch Filter Frequency Index (continued on next page)

NOTCH cont.

FREQ (Hz)	FREQ Index	FREQ (Hz)	FREQ Index
810	80	1210	120
820	81	1220	121
830	82	1230	122
840	83	1240	123
850	84	1250	124
860	85	1260	125
870	86	1270	126
880	87	1280	127
890	88	1290	128
900	89	1300	129
910	90	1310	130
920	91	1320	131
930	92	1330	132
940	93	1340	133
950	94	1350	134
960	95	1360	135
970	96	1370	136
980	97	1380	137
990	98	1390	138
1000	99	1400	139
1010	100	1410	140
1020	101	1420	141
1030	102	1430	142
1040	103	1440	143
1050	104	1450	144
1060	105	1460	145
1070	106	1470	146
1080	107	1480	147
1090	108	1490	148
1100	109	1500	149
1110	110	1510	150
1120	111	1520	151
1130	112	1530	152
1140	113	1540	153
1150	114	1550	154
1160	115	1560	155
1170	116	1570	156
1180	117	1580	157
1190	118	1590	158
1200	119	1600	159

Notch Filter Frequency Index (continued on next page)

NOTCH cont.

FREQ (Hz)	FREQ Index
1610	160
1620	161
1630	162
1640	163
1650	164

Notch Filter Frequency Index

Quality Factor	Quality Index
1	0
2	1

Notch Filter Quality Factor Index

OFFSET

FUNCTION Amplifier Offset Cancellation

SYNTAX OFFSET(n)

RTC CODE 5Fh

ARGUMENTS

n a single byte, bit coding the ONLY axis involved.

DESCRIPTION

This command minimizes the offset generated by the D/A converter. Upon completion of offset tuning, an interrupt is generated to the host. The condition is recorded in DPR interrupt status register location 009h. The DPR status register location 00Ch will identify the axis responsible. Bit 6 of DPR locations 3FEh, 7FEh is also set.

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.
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 to the point that position error is forced to zero. In 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 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 ( )      ;make sure system can stop
CTRL ( )        ;set gains
KILIMIT ( )     ;put system in a position loop, make sure integral
                ;gain is non-zero
.
.
OFFSET ( )
```

OFFSET cont.

EXAMPLE

After verifying that OFFSET Steps 1-3 (see DESCRIPTION, above) have been followed, do offset tuning for axis 3.

The value of the RTC argument is:

n : 04h

OUTGAIN

FUNCTION Output Loop Gain

SYNTAX OUTGAIN(n, m_1, ... , m_4)

RTC CODE 81h

ARGUMENTS

n a single byte, bit coding the axes involved.
m_x a single byte to specify the gains.

m=0	gain=1
m=1	gain=2
m=2	gain=4
m=3	gain=8
m=4	gain=16

DESCRIPTION

This command is used to set the gain for the output of the position loops. The default m is set to 0 (gain = 1).

Note: Please see block diagram on Page 4-19.

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 low gain. This may be compensated for by a loop gain adjustment. In practice, this command may use an argument greater than one if the encoder line number is less than 1000.

OUTGAIN cont.

Command Sequence Example

```
MAXACC ( )      ;make sure system can stop
CTRL ( )        ;set gains
KILIMIT ( )
OUTGAIN ( )
```

EXAMPLE

Program output loop gains of eight for axis 3 and two for axis 4.

The values of the RTC arguments are:

n	=	0Ch
m_3	=	03h
m_4	=	01h

OUTREL

FUNCTION Output Relay

SYNTAX OUTREL(n, rl_1, ... , rl_4)

RTC CODE 59h

ARGUMENTS

n — a single byte, bit coding the axes involved.

rl_x — a single byte, bit mapping the outputs for axis x.

bit=0	active-LOW output
bit=1	active-HIGH output

bit 7	unused
bit 6	unused
bit 5	unused
bit 4	unused
bit 3	unused
bit 2	OUT2 output for axis x general purpose output 3
bit 1	OUT1 output for axis x general purpose output 2
bit 0	OUT0 output for axis x general purpose output 1

Note: The general purpose outputs (1-3) are mapped to axis 4 (i.e., n=08h).

DESCRIPTION

This command allows the status of all outputs to be set.

SEE ALSO DISABORT, ENABORT

OUTREL cont.

APPLICATION

This command can be used for general purpose logical output operation.

Command Sequence Example

No preparation is required before running this instruction.

EXAMPLE

Generate an active-HIGH signal on general purpose output 1.

The arguments for this instruction will be:

n	:	08h
rl_3	:	01h

PARREAD

FUNCTION Parameter Readback

SYNTAX PARREAD(m)

RTC CODE 5Eh

ARGUMENTS

m a byte which indicates the parameters to echo.

m=10h	axis 1 position loop gain values
m=11h	axis 2 position loop gain values
m=12h	axis 3 position loop gain values
m=13h	axis 4 position loop gain values
m=14h	K_i limit value
m=15h	position loop output gain values
m=16h	maximum acceleration
m=17h	enabled interrupt
m=18h	mode of operation
m=19h	following error and halt interrupt setpoint
m=1Ah	following error interrupt setpoint
m=1Bh	axis 1 and 2 position breakpoint interrupt setpoint
m=1Ch	axis 3 and 4 position breakpoint interrupt setpoint
m=1Dh	buffer breakpoint interrupt setpoint and contouring block transfer rate
m=1Eh	enabled limit switch interrupt
m=1Fh	multi-turn base values
m=20h	abort maximum acceleration
m=21h	master/slave status
m=22h	output relay status
m=23h	logic state of inputs

PARREAD cont.

DESCRIPTION

Upon the execution of this command, Mx4 cnC++ echoes the desired parameters to DPR locations 0B8h - 0BFh. "m" is echoed to DPR location 0B7h if the parameters are ready in the DPR. Parameters may take more than 5ms to echo back to the DPR. Host can use the following algorithm:

1. write m to DPR location 3C3h
2. write 0 to DPR location 0B7h
3. write RTC command code to DPR location 3C2h
4. poll DPR location 0B7h until m is echoed
5. read the data from DPR location 0B8h - 0BFh

DATA FORMAT

For each type of parameter, DPR locations 0B8h - 0BFh are interpreted differently. The following shows the format for each type of parameter:

1. Position loop gains (m=10h - m=13h)

0B8h	K_i low byte
0B9h	K_i high byte
0BAh	K_p low byte
0BBh	K_p high byte
0BCh	K_f low byte
0BDh	K_f high byte
0BEh	K_d low byte
0BFh	K_d high byte

PARREAD cont.

2. K_i limit (m=14h)

0B8h	K_i limit for axis 1
0B9h	K_i limit for axis 2
0BAh	K_i limit for axis 3
0BBh	K_i limit for axis 4
0BCh	
:	not used
0BFh	

Note: $0 \leq \text{Kilimit} \leq 14$

3. Position loop output gain (m=15h)

0B8h	m specified gains for axis 1
0B9h	m specified gains for axis 2
0BAh	m specified gains for axis 3
0BBh	m specified gains for axis 4
0BCh	
:	not used
0BFh	

Note: $0 \leq m \leq 4$

4. Maximum acceleration (m=16h)

0B8h	low byte acceleration for axis 1
0B9h	high byte acceleration for axis 1
0BAh	low byte acceleration for axis 2
0BBh	high byte acceleration for axis 2
0BCh	low byte acceleration for axis 3
0BDh	high byte acceleration for axis 3
0BEh	low byte acceleration for axis 4
0BFh	high byte acceleration for axis 4

PARREAD cont.

5. Enabled interrupt (m=17h)

0B8h	bit 0 codes buffer breakpoint interrupt
0B9h	low nibble bit codes the following error and halt interrupts, high nibble bit codes the following error interrupts
0BAh	low nibble bit codes the index pulse interrupts, high nibble bit codes the position breakpoint interrupts
0BBh	low nibble bit codes the motion complete interrupts, high nibble bit codes the probe interrupts
0BCh	low nibble bit codes the positive feedback interrupts, high nibble bit codes the encoder lost interrupts
0BDh : 0BFh	not used

6. Mode of operation (m=18h)

0B8h	low nibble bit codes the axes in axis move operation
0B9h	low nibble bit codes the axes in stop operation
0BAh	low nibble bit codes the axes in velmode operation
0BBh	low nibble bit codes the axes in contouring operation
0BCh : 0BFh	not used

PARREAD cont.

7. Following error and halt interrupt setpoint (m=19h)

0B8h	low byte setpoint for axis 1
0B9h	high byte setpoint for axis 1
0BAh	low byte setpoint for axis 2
0BBh	high byte setpoint for axis 2
0BCh	low byte setpoint for axis 3
0BDh	high byte setpoint for axis 3
0BEh	low byte setpoint for axis 4
0BFh	high byte setpoint for axis 4

8. Following error interrupt setpoint (m=1Ah)

0B8h	low byte setpoint for axis 1
0B9h	high byte setpoint for axis 1
0BAh	low byte setpoint for axis 2
0BBh	high byte setpoint for axis 2
0BCh	low byte setpoint for axis 3
0BDh	high byte setpoint for axis 3
0BEh	low byte setpoint for axis 4
0BFh	high byte setpoint for axis 4

9. Position breakpoint setpoint (m=1B for axes 1 and 2, 1Ch for axes 3 and 4)

0B8h	low word low byte setpoint for axis 1 or 3
0B9h	low word high byte setpoint for axis 1 or 3
0BAh	high word low byte setpoint for axis 1 or 3
0BBh	high word high byte setpoint for axis 1 or 3
0BCh	low word low byte setpoint for axis 2 or 4
0BDh	low word high byte setpoint for axis 2 or 4
0BEh	high word low byte setpoint for axis 2 or 4
0BFh	high word high byte setpoint for axis 2 or 4

PARREAD cont.

10. Buffer breakpoint interrupt setpoint and contouring block transfer rate (m=1Dh)

0B8h	buffer breakpoint interrupt setpoint		
0B9h	= 00h	:	2nd order contouring
	= FFh	:	cubic spline contouring
0BAh	low byte, block transfer rate		
0BBh	high byte, block transfer rate (for cubic spline only)		
0BCh : 0BFh	not used		

11. Enabled limit switch interrupt (m=1Eh)

(bit=1 indicates the corresponding interrupt is enabled.)

0B8h	echo m byte of ENABORT RTC for axis 1
0B9h	byte bit-mapping the input enables for axis 1
0BAh	echo m byte of ENABORT RTC for axis 2
0BBh	byte bit-mapping the input enables for axis 2
0BCh	echo m byte of ENABORT RTC for axis 3
0BDh	byte bit-mapping the input enables for axis 3
0BEh	echo m byte of ENABORT RTC for axis 4
0BFh	byte bit-mapping the input enables for axis 4

12. Multi-turn base values (m=1Fh)

0B8h	low byte base value for axis 1
0B9h	high byte base value for axis 1
0BAh	low byte base value for axis 2
0BBh	high byte base value for axis 2
0BCh	low byte base value for axis 3
0BDh	high byte base value for axis 3
0BEh	low byte base value for axis 4
0BFh	high byte base value for axis 4

PARREAD cont.

13. Abort maximum acceleration (m=20h)

0B8h	low byte acceleration for axis 1
0B9h	high byte acceleration for axis 1
0BAh	low byte acceleration for axis 2
0BBh	high byte acceleration for axis 2
0BCh	low byte acceleration for axis 3
0BDh	high byte acceleration for axis 3
0BEh	low byte acceleration for axis 4
0BFh	high byte acceleration for axis 4

14. Master/Slave status (m=21h)

0B8h	=00h, configured as Master =11h, configured as Slave
0B9h : 0BFh	not used

15. Output relay status (m=22h)

0B8h	low nibble	:	bit codes OUT0
	high nibble	:	bit codes OUT1
0B9h	not used		
0BAh	bit 7	:	OUT3(4)
	bit 6	:	OUT4(4)
	bit 5-4	:	not used
	bit 3	:	OUT2(4)
	bit 2-0	:	not used
0BBh	not used		
0BCh	bit 7	:	RL2
	bit 6	:	RL1
	bit 5-0	:	not used
0BDh : 0BFh	not used		

PARREAD cont.

16. Logic state of inputs (m=23h)

0B8h	not used		
0B9h	echo inp_1 byte of INPSTATE RTC		
0BAh	not used		
0BBh	echo inp_2 byte of INPSTATE RTC		
0BCh	not used		
0BDh	bit 7	:	echo bit 5 of inp_3 byte of INPSTATE RTC
	bit 6	:	echo bit 4 of inp_3 byte of INPSTATE RTC
	bit 0-5	:	not used
0BEh	not used		
0Bfh	not used		

SEE ALSO none

APPLICATION

This command can be used as a diagnostic tool to monitor all system parameters.

Command Sequence Example

No preparation is required before running this instruction.

EXAMPLE

Verify the gains settings for axis 2 by instructing Mx4 cnC++ to echo the values to the DPR with a PARREAD command.

The value of the RTC argument is:

m = 11h

POSBRK

FUNCTION Position Breakpoint Interrupt

SYNTAX POSBRK(n, pos_1, ... , pos_4)

RTC CODE 6Bh

ARGUMENTS

n	a single byte, bit coding the axes involved.
pos_x	32 bit 2's complement position breakpoint value for axis x.

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 locations 3FEh, 7EFh is also set.

Note: The position breakpoint is calculated as an absolute distance from the present position (position at the moment at which the 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.

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

Note: HOME and HOMESFT commands will automatically disable the position breakpoint interrupt. The user is responsible to re-enable the interrupt again.

SEE ALSO DISABL, HOME, HOMESFT

POSBRK cont.

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 ( )      ;make sure system can stop
CTRL ( )        ;set gains
KILIMIT ( )
OUTGAIN ( )
```

EXAMPLE

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

The values of the RTC arguments are:

n	:	03h
pos_1	:	0000EA60h
pos_2	:	0007A120h

POSFEED

FUNCTION Enable Positive Feedback Interrupt

SYNTAX POSFEED(n)

RTC CODE 5Bh

ARGUMENTS

n a single byte, bit coding the axes involved.

DESCRIPTION

This command enables the positive feedback loop interrupt for the specified axes. Positive feedback interrupt is generated if the following conditions are met (for the axis in question):

1. following error is > 2000 counts
2. one of the possible error cases listed below is met
3. the above two conditions hold for 0.3 seconds

Possible Error Cases:

A. command position values increasing, actual position values decreasing

B. command position values decreasing, actual position values increasing

Note: The following cases are allowed due to possible friction-related motion characteristics:

A. command position values increasing, actual position values unchanged

B. command position values decreasing, actual position values increasing

POSFEED cont.

The DPR interrupt status locations 009h and 00Ah record the occurrence and source of this interrupt. Bit 6 of DPR locations 3FEh, 7EFh is also set.

SEE ALSO DISABL2

APPLICATION

A necessary diagnostic feature for all servo control applications.

Command Sequence Example

No preparation is required before running this instruction.

EXAMPLE

Enable the positive feedback loop interrupt for all four axes.

The value of the RTC argument is:

n : 0Fh

PRBINT

FUNCTION General Purpose Interrupt

SYNTAX PRBINT(n, m)

RTC CODE 6Ch

ARGUMENTS

n a single byte, bit coding the \PRx echoed back to the DPR.
m byte which indicates the ONLY source of the interrupt signal.

m=1 : from \PR0
m=2 : from \PR1

DESCRIPTION

Upon the execution of this command, the servo control card will search for the first \PRx pulse edge. The pulse edge results in the generation of an interrupt, stop ALL the axes and registration of the actual position for all axes in DPR location 0A7h-0B6h. (The hand shaking bytes are 0C8h and 0D0h for Mx4 cnC++ and host respectively.) An interrupt is generated after an axis is stopped. The DPR interrupt status register locations 000h and 006h record the occurrence and source (echo of values n and m) of this interrupt. Bit 1 of DPR locations 3FEh, 7EFh is also set.

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

Note: General purpose probe interrupt and index pulse CAN BE ENABLED simultaneously.

Note: Abort maximum acceleration must be set non-zero.

PRBINT cont.

SEE ALSO DISABL, ABORTACC

APPLICATION

This instruction is useful in probing applications. Since PRBINT registers all positions when an interrupt occurs (falling pulse edge is detected) and brings all axes to a stop, it can be used in accurate recording of surface dimensions by a probe.

Command Sequence Example

```
ABORTACC ( )    ;make sure that system can be stopped
CTRL ( ) ;these instructions enable system to stop motion
KILIMIT ( )
.
.
PRBINT ( )
```

EXAMPLE

Enable the \PR2 probe interrupt.

The values of the RTC arguments are:

n	:	02h
m	:	02h

RESET

FUNCTION Reset MX4

SYNTAX RESET(AAh, AAh)

RTC CODE 72h

ARGUMENTS

AAh reset signature byte.

DESCRIPTION

This command brings the servo controller card back to power-up state. Upon Mx4 cnC++'s reset completion, a host interrupt is generated via bit 4 of DPR locations 3FEh, 7FEh.

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 cnC++ controller card.

The arguments of RESET are AAh, AAh (2 bytes).

START

FUNCTION Start Contouring Motion

SYNTAX START(n)

RTC CODE 6Dh

ARGUMENTS

n a single byte, bit coding the axes involved.

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 RTC 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 gains
KILIMIT ( )
BTRATE ( )          ;set block transfer rate
BBINT ( )           ;set the breakpoint in the ring buffer
.
.
START ( )           ;start contouring
```

EXAMPLE

Start contouring motion in axes 2 and 3.

The values of the RTC argument is:

n : 06h

STOP

FUNCTION Stop Motion

SYNTAX STOP(n)

RTC CODE 6Eh

ARGUMENTS

n a single byte, bit coding the axes involved.

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 this command 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: An emergency stop signal, ESTOP/, 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: STOP command 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 via a previously executed STOP RTC or active ESTOP input, MX4 will ignore any motion commands (AXMOVE, START or VELMODE) and will report an "RTC Command Ignored" interrupt to the host if such a command is received by Mx4 cnC++ for an axis that is not yet halted. The above motion commands should not be sent to Mx4 cnC++ for a halting axis until the axis motion has come to a stop.

STOP cont.

SEE ALSO AXMOVE, MAXACC, START

APPLICATION

For all applications involving bringing speed to zero (0) in the quickest possible manner.

Command Sequence Example

```
MAXACC ( )      ;make sure system can stop
CTRL ( ) ;set gains
KILIMIT ( )
BTRATE ( )      ;set block transfer
BBINT ( )       ;set the breakpoint in the ring buffer
.
.
STOP ( ) ;stop the motion
.               ;upon completion of stop (command) trajectory
.               ;Mx4 cnC++ generates motion complete interrupt
```

EXAMPLE

Bring the motion of axes 1 and 4 to a halt.

The value of the RTC argument is:

n : 09h

SYNC

FUNCTION Master / Slave Select

SYNTAX SYNC(m)

RTC CODE 87h

ARGUMENTS

m a byte that selects the Master / Slave status of the Mx4 cnC++ board.

m=0 : Mx4 cnC++ is configured as a Master
m≠0 : Mx4 cnC++ is configured as a Slave

DESCRIPTION

If more than one Mx4 cnC++ card is to be used in a system and card-to-card synchronization is required, the SYNC RTC should be used. The SYNC RTC allows multiple Mx4 cnC++ cards to operate in synchronization within a system by specifying a single Master and the remaining card(s) as Slaves. SYNC establishes the Mx4 cnC++ card as either a Master or Slave in a multiple card system. If only one (1) Mx4 cnC++ is used in a host computer system, that Mx4 cnC++ must be configured as a Master.

Note: Mx4 cnC++ powers-up and resets to a default Master status.

In addition to configuring the Mx4 cnC++ cards via the SYNC RTC (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 cnC++ connects to the SLAVE signal of each of the Slave Mx4 cnC++(s).

SYNC cont.

SEE ALSO none

APPLICATION

This command is used in applications where tight coordination of more than four axes is required. This command essentially slaves several MX4 cnC++ MB-IO cards to a single master MX4 cnC++ MB-IO. 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

To initialize a card as Slave

m : 01h

VECCHG

FUNCTION 2nd Order Contouring Vector Change

SYNTAX VECCHG(n, m)

RTC CODE 6Fh

ARGUMENTS

n	a single byte, bit coding the axes involved.
m	8 bit positive 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: 3 buffer levels are used to implement this instruction.

SEE ALSO START

APPLICATION

See START.

VECCHG cont.

Command Sequence Example

```
MAXACC ( )      ;make sure system can stop
CTRL ( )        ;set gains
KILIMIT ( )
BTRATE ( )      ;set the block transfer rate
BBINT ( )       ;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 contouring in axes 1, 2, and 3 after the 23rd segment move command of the ring buffer.

The values of the RTC arguments are:

n : 07h
m : 17h

VELMODE

FUNCTION Velocity Mode

SYNTAX VELMODE(n, vel_1, ... , vel_4)

RTC CODE 70h

ARGUMENTS

n	a single byte, bit coding the axes involved.
vel_x	32 bit 2's complement velocity value for axis x.

Note: Velocity is represented by a 27 bit 2's complement number which is sign extended to 32 bits. Velocity is partitioned as 16 bits integer, 16 bits fraction.

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. Velocity mode uses the maximum acceleration / deceleration value to accelerate or decelerate to the desired velocity.

Note: VELMODE command will be ignored if the maximum acceleration / deceleration is equal to zero (e.g. MAXACC not issued).

Note: In order to obtain units of encoder edge counts / 200µsec, the host must use a division factor as specified:

Mx4 cnC++	816
VECTOR4 installed	840

SEE ALSO MAXACC

VELMODE cont.

APPLICATIONS

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 ( )      ;make sure system can stop
CTRL ( ) ;set gains
KILIMIT ( )
.
.
VELMODE ( )
```

EXAMPLE

Engage axis 2 in velocity mode with a velocity of 3.71 counts/200µsec.

The values of the RTC arguments are:

n	:	02h
vel_2	:	0003B5C3h

6 Mx4 cnC++ Host-Based Programming

Mx4 cnC++ - Host Communication

The host communicates with the PC/AT Mx4 cnC++ through the host computer ISA bus. The communication takes place across a Dual Port RAM (DPR) buffer on the Mx4 cnC++ card (see Fig. 6-1). Through this buffer, the host may read system state variables such as position and velocity, interrupt internal Mx4 cnC++ parameters, write real time instructions to the Mx4 cnC++ card, monitor the interrupt status of Mx4 cnC++ and much more.

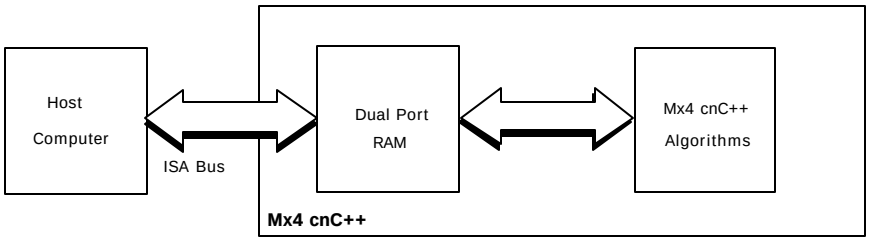


Fig. 6-1: Host - Mx4 cnC++ Dual Port RAM Interface

Host - Mx4 cnC++ Interface

The host communicates with Mx4 cnC++ via a 2048 byte Dual Port RAM (DPR_. This DPR is functionally split into seven blocks which are described below in Table 6-1.

DPR BLOCK	ADDRESS RANGE	DESCRIPTION
Status Registers	000h - 08Dh	The status register block includes Mx4 cnC++ card status codes as well as interrupt source information.
Hardware Signature	08Eh - 093h	These bytes code, in ASCII, the hardware platform (PC/AT, Multibus or VME), hardware options (with I/O or standard configuration), and the board's revision number.
Parameter Updates	094- 114h	System parameters such as actual position, following error and actual velocity are available for the host to read in this block.
Signature Window	115h - 11Fh	The Mx4 cnC++ card writes a signature using ASCII codes to the signature window at power-up. The host may check for this signature to verify installation of a Mx4 cnC++ card.
2nd Order Contouring Ring Buffer	120h - 3C1h	This block of the DPR is reserved for 2nd order contouring motion or coordinated move data points. The host downloads data to Mx4 cnC++ via this "ring buffer".
Cubic Spline Contouring Ring Buffer	400h - 7F1h	This block of the DPR is reserved for cubic spline contouring motion data points. The host downloads data to Mx4 cnC++ via this “ring buffer”.
RTC Window	3FCh - 3FFh 7FEh, 7FFh	This window in the DPR serves as a buffer for the host to send RTCs to Mx4 cnC++.
Interrupt Registers	3FCh - 7FFh	This block is used in the setting and re-setting of hardware interrupts to the host.

Table 6-1: Mx4 cnC++ Dual Port RAM Blocks

Communication Protocols

In order to maintain a healthy communication interface between the host and Mx4 cnC++, some simple communication protocols must be adhered to by the host.

Each location in the DPR is labeled (see *Mx4 cnC++ Dual Port RAM Organization*) with two read/write access codes. One code for the host, one for Mx4 cnC++. The access codes are:

RO	:	read only access
WO	:	write only access
RW	:	read and write access

These "restrictions" are enforced only by convention in the host and Mx4 cnC++ software. They are included to help the user understand how the various locations in the DPR are used by both the host and Mx4 cnC++.

Much of the data in the DPR that the host and Mx4 cnC++ must read or write is multi-byte data (such as 32-bit actual position values) and thus requires multi-address accesses to the DPR. In order to ensure that multi-byte values are not corrupted by unsynchronized accesses, many DPR 'windows' are protected via 'access bytes'. See Fig. 6-3.

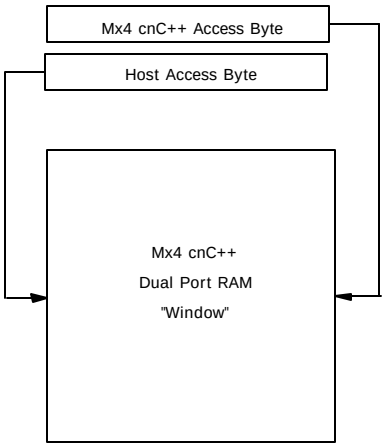


Fig. 6-3: Mx4 cnC++ Dual Port RAM Access Bytes

Each 'window' includes both a host access byte and a Mx4 cnC++ access byte which are used to control access to the window. A more detailed explanation of how to use these bytes is offered in the *Mx4 cnC++ Dual Port RAM Organization* section and will be discussed further in *Communication Protocols Revisited*.

Mx4 cnC++ Dual Port RAM Organization

The 2 Kbyte DPR is partitioned as follows in Table 6. (The names given to individual bytes and groups of data are for reference only.)

Status Registers (locations 000h - 08Dh)

NAME	ADDRESS	ACCESS		DESCRIPTION
		DSP	HOST	
The status register bytes indicate the interrupt status of Mx4 cnC++. Mx4 cnC++ updates or writes to these locations in an OR fashion. Thus, Mx4 cnC++ does not reset interrupt status bits. The host, after reading or recognizing an interrupt status register must reset bits at its own discretion.				
DSPSTAT1	000h	RW	RW	Mx4 cnC++ status register for interrupts. Polled by the host to determine internal Mx4 cnC++ status. bit 0: following error halt and interrupt bit 1: following error interrupt bit 2: index pulse interrupt bit 3: position breakpoint interrupt bit 4: motion complete interrupt bit 5: probe signal interrupt bit 6: conflicting commands (ignore the new motion-related command and send an interrupt) bit 7: RTC command is ignored because STOP is in progress

Table 6: Dual Port RAM Status Registers (continued on next page)

NAME	ADDRESS	ACCESS		DESCRIPTION
		DSP	HOST	
DSPSTAT2	009h	RW	RW	Mx4 cnC++ status register for register for to determine internal Mx4 cnC++. status. bit 0: positive feedback interrupt bit 1: encoder lost interrupt bit 2: offset cancel finished bit 3: inputs (limit switch/fault) interrupt bits 4-7: not used
The INTAXIS bytes code the source(s) of the interrupt(s) by setting a bit(s) (unless otherwise noted): bit 0: axis 1 bit 1: axis 2 bit 2: axis 3 bit 3: axis 4 bits 4-7: not used				
INTAXIS	001h	RW	RW	source of following error and halt interrupt
INTAXIS	002h	RW	RW	source of following error interrupt
INTAXIS	003h	RW	RW	source of index pulse interrupt
INTAXIS	004h	RW	RW	source of position breakpoint interrupt
INTAXIS	005h	RW	RW	source of motion complete interrupt
INTAXIS	006h	RW	RW	source of probe signal interrupt low nibble : echo m high nibble : echo n
INTAXIS	007h	RW	RW	source of conflicting commands interrupt
INTAXIS	008h	RW	RW	source of RTC command ignored because STOP is in progress
INTAXIS	00Ah	RW	RW	source of positive feedback interrupt
INTAXIS	00Bh	RW	RW	source of encoder loss interrupt
INTAXIS	00Ch	RW	RW	source of offset cancel finished
INTAXIS	00Dh	RW	RW	source of limit switch/fault interrupt

reserved	00Eh-08Dh	-	-	unused locations

Table 6 Cont.: Dual Port RAM Status Registers

Hardware Signature Window (locations 08Eh - 093h)

NAME	ADDRESS	ACCESS		DESCRIPTION
		DSP	HOST	
SIGNATURE	08Eh	WO	RO	Bus designator byte: ASCII "P" : PC/AT ASCII "M" : Multibus ASCII "V" : VME
SIGNATURE	08Fh	WO	RO	Hardware option byte: ASCII "I" : I/O integer 0 : standard configuration
SIGNATURE	090h	WO	RO	Revision byte: ASCII "A" : revision A ASCII "B" : revision B etc.
SIGNATURE	091h - 093h	WO	RO	reserved for future options ... currently unused

Fig. 7: Dual Port RAM Hardware Signature Window

Parameter Updates (locations 094h - 114h)

NAME	ADDRESS	ACCESS		DESCRIPTION
		DSP	HOST	
LIMSW	094h	WO	RO	limit switch 0, 1 status (real time) A set bit indicates active: bit 0: axis 1 + O.T. bit 1: axis 2 + O.T. bit 2: axis 3 + O.T. bit 3: axis 4 + O.T. bit 4: axis 1 - O.T. bit 5: axis 2 - O.T. bit 6: axis 3 - O.T. bit 7: axis 4 - O.T.
reserved	095h	-	-	unused location
FAULT	096h	WO	RO	fault status (real time). A set bit indicates active: bit 0-3: not used bit 4: Gen. Purpose Input 4 bit 5: Gen. Purpose Input 5 bits 6-7: not used
MFRAC1	097h - 098h	WO	RO	axis 1 multi-turn fraction LSB, MSB (16 bit two's complement)
MTURN1	099h - 09Ah	WO	RO	axis 1 multi-turn # turns LSB, MSB (16 but two's complement)
MFRAC2	09Bh - 09Ch	WO	RO	axis 2 multi-turn fraction LSB, MSB (16 bit two's complement)
MTURN2	09Dh - 09Eh	WO	RO	axis 2 multi-turn # turns LSB, MSB (16 but two's complement)
MFRAC3	09Fh -0A0h	WO	RO	axis 3 multi-turn fraction LSB, MSB (16 bit two's complement)
MTURN3	0A1h - 0A2h	WO	RO	axis 3 multi-turn # turns LSB, MSB (16 but two's complement)
MFRAC4	0A3h - 0A4h	WO	RO	axis 4 multi-turn fraction LSB, MSB (16 bit two's complement)
MTURN4	0A5h - 0A6h	WO	RO	axis 4 multi-turn # turns LSB, MSB (16 but two's complement)
PRB10-3	0A7h - 0AAh	WO	RO	axis 1 probe interrupt position LSB,...,MSB (32 bit two's complement)
PRB20-3	0ABh - 0AEh	WO	RO	axis 2 probe interrupt position LSB,...,MSB (32 bit two's complement)

Table 8: Dual Port RAM Parameter Updates (cont. on next page)

PRB30-3	0AFh - 0B2h	WO	RO	axis 3 probe interrupt position LSB,...,MSB (32 bit two's complement)
PRB40-3	0B3h - 0B6h	WO	RO	axis 4 probe interrupt position LSB,...,MSB (32 bit two's complement)
PARACC	0B7h	WO	RW	Parameter readback window m echo
PARRDBK	0B8h - 0BFh	WO	RO	Parameter readback window (see PARREAD RTC description)
reserved	0C0h - 0C2h	-	-	unused locations
e M4ACC bytes are used as access flags. When the Mx4 cnC++ needs to access a parameter update window, it sets the corresponding M4ACC byte to 01h. The host must test these flags to see if values can be written to or read from the parameter window in question. M4ACC=00h : Mx4 cnC++ is not using window. Host may set corresponding HOSTACC=01h and may access window. M4ACC=01h : Mx4 cnC++ is using window. Host must wait until this byte is cleared before accessing the window in question.				
M4ACC	0C3h	WO	RO	Mx4 cnC++ is using 0D3h - 0E2h window
M4ACC	0C4h	WO	RO	Mx4 cnC++ is using 0E3h - 0F2h window
M4ACC	0C5h	WO	RO	Mx4 cnC++ is using 0F3h - 102h window
M4ACC	0C6h	WO	RO	Mx4 cnC++ is using 103h - 112h window
M4ACC	0C7h	WO	RO	Mx4 cnC++ is using 113h - 114h window
M4ACC	0C8h	WO	RO	Mx4 cnC++ is using 0A7h - 0B6h window
M4ACC	0C9h	WO	RO	Mx4 cnC++ is using 097h - 0A6h window
M4ACC	0CAh	WO	RO	unused location
he HOSTACC bytes are used as access flags. When the host needs to access a parameter update window, it sets the corresponding HOSTACC byte to 01h. The Mx4 cnC++ will test these flags to see if values can be written to or read from the parameter window in question. HOSTACC=00h : Host is not using window. Mx4 cnC++ may set corresponding M4ACC=01h and may access window. HOSTACC=01h : Host is using window. Mx4 cnC++ must wait until this byte is cleared before accessing the window in question.				

Table 8 cont.: Dual Port RAM Parameter Updates (cont. on next page)

HOSTACC	0CBh	RO	RW	host is using 0D3h - 0E2h window
HOSTACC	0CCh	RO	RW	host is using 0E3h - 0F2h window
HOSTACC	0CDh	RO	RW	host is using 0F3h - 102h window
HOSTACC	0CEh	RO	RW	host is using 103h - 112h window
HOSTACC	0CFh	RO	RW	host is using 113h - 114h window
HOSTACC	0D0h	RO	RW	host is using 0A7h - 0B6h window
HOSTACC	0D1h	RO	RW	host is using 097h - 0A6h window
HOSTACC	0D2h	RO	RW	unused location
See Chapter 4's *Modes of Operation, State Variables* for details concerning the format of position, velocity and following error data.				
POS10-3	0D3h - 0D6h	WO	RO	axis 1 position LSB,...,MSB (32 bit two's complete)
POS20-3	0D7h - 0DAh	WO	RO	axis 2 position LSB,...,MSB (32 bit two's complete)
POS30-3	0DBh - 0DEh	WO	RO	axis 3 position LSB,...,MSB (32 bit two's complete)
POS40-3	0DFh - 0E2h	WO	RO	axis 4 position LSB,...,MSB (32 bit two's complete)
In order to obtain units of encoder edge counts/sampling period for velocity, the host must use a division factor (as specified in the VELMODE RTC description).				
VEL10-3	0E3h - 0E6h	WO	RO	axis 1 velocity LSB,...,MSB (32 bit two's complement)
VEL20-3	0E7h - 0EAh	WO	RO	axis 2 velocity LSB,...,MSB (32 bit two's complement)
VEL30-3	0EBh - 0EEh	WO	RO	axis 3 velocity LSB,...,MSB (32 bit two's complement)
VEL40-3	0EFh - 0F2h	WO	RO	axis 4 velocity LSB,...,MSB (32 bit two's complement)
FE10-3	0F3h - 0F6h	WO	RO	axis 1 following error LSB,...MSB (32 bit two's complement)

Table 8 cont.: Dual Port RAM Parameter Updates (cont. on next page)

FE20-3	0F7h - 0FAh	WO	RO	axis 2 following error LSB,...MSB (32 bit two's complement)
FE30-3	0FBh - 0FEh	WO	RO	axis 3 following error LSB,...MSB (32 bit two's complement)
FE40-3	0FFh - 102h	WO	RO	axis 4 following error LSB,...MSB (32 bit two's complement)
IND10-3	103h - 106h	WO	RO	axis 1 index position LSB,...MSB (32 bit two's complement)
IND20-3	107h - 10Ah	WO	RO	axis 2 index position LSB,...MSB (32 bit two's complement)
IND30-3	10Bh - 10Eh	WO	RO	axis 3 index position LSB,...MSB (32 bit two's complement)
IND40-3	10Fh - 112h	WO	RO	axis 4 index position LSB,...MSB (32 bit two's complement)
ENCSTAT MARKER	113h	WO	RO	Status of encoders. A set bit indicates a failure of the corresponding hardware item: bit 0: axis 1 encoder bit 1: axis 2 encoder bit 2: axis 3 encoder bit 3: axis 4 encoder Real time marker reporting (a set bit indicates index pulse) bit 4 : axis 1 index pulse bit 5 : axis 2 index pulse bit 6 : axis 3 index pulse bit 7 : axis 4 index pulse
SERVOCHK	114h	RW	RO	Servo check byte. Bit 0 is set if Mx4 cnC++ DSP internal stack is overflowed. Then a system reset may be necessary.

Table 8 Cont.: Dual Port RAM Parameter Updates

Signature Window (locations 115h - 11Fh)

NAME	ADDRESS	ACCESS		DESCRIPTION
		DSP	HOST	
The SIGNATURE bytes contain the ASCII code controller card signature. The signature will be present if the card is operation correctly.				
SIGNATURE	115h	WO	RO	ASCII "M"
SIGNATURE	116h	WO	RO	ASCII "X"
SIGNATURE	117h	WO	RO	ASCII "4"
SIGNATURE	118h	WO	RO	integer part of dsp1 software version number
SIGNATURE	119h	WO	RO	decimal part of dsp1 software version number
SIGNATURE	11Ah	WO	RO	ASCII "+"
SIGNATURE	11Bh	WO	RO	integer part of dsp2 software version number
SIGNATURE	11Ch	WO	RO	decimal part of dsp2 software version number
SIGNATURE	11Dh	WO	RO	ASCII "+"
SIGNATURE	11Eh	WO	RO	integer part of VECTOR4 version number
SIGNATURE	11Fh	WO	RO	decimal part of VECTOR4 version number
If VECTOR4 is not installed, DSP location 11Dh - 11Fh will be zero.				

Table 9: Dual Port RAM Signature Window

2nd Order Contouring Ring Buffer (locations 120h - 3C1h)

NAME	ADDRESS	ACCESS		DESCRIPTION
		DSP	HOST	
RINGBUF	120h - 3BFh	RO	WO	672 byte ring buffer which contains host contouring commands (4 bytes position, 4 bytes velocity) to be processed by the DSP .. The length of all messages deposited by the host in this buffer must be a multiple of 8 bytes.
INPTR	3C0h	RO	RW	Pointer to the next free location in RINGBUF that the host will write to (expressed as an offset from the start of the buffer in multiples of 8 bytes).
OUTPTR	3C1h	RW	RO	Pointer to the next location in RINGBUF that Mx4 cnC++ will read from (expressed as an offset from the start of the buffer in multiples of 8 bytes).

Table 10: Dual Port RAM Ring Buffer

Real Time Command (RTC) (locations 3C2h - 3FBh)

NAME	ADDRESS	ACCESS		DESCRIPTION
		DSP	HOST	
RTC	3C2h	RW	RW	Real time command byte. If this byte is non-zero, Mx4 cnC++ will interpret it as a command and execute it, using the following locations as its arguments. When execution is complete and Mx4 cnC++ is ready for the next command, RTC will be set to zero.
ARGMNTS	3C3h - 3FBh	RO	WO	57 bytes of argument storage area. The usage of this area depends on the real time command to be executed. The host must set up the argument area correctly before writing a command code to RTC, to ensure that Mx4 cnC++ reads the arguments properly.

Table 11: Dual Port RAM Real Time Command (RTC)

Interrupt Registers (locations 3FCh - 3FFh, 7FEh, 7FFh)

NAME	ADDRESS	ACCESS		DESCRIPTION
		DSP	HOST	
MINTACC	3FCh	RW	RO	A Mx4 cnC++ access flag byte. When Mx4 cnC++ needs to access location 3FEh, 7FEh or the status registers window (000h - 08Dh), it sets this byte equal to 01h. The host must test this flag to see if values can be written to or read from the window.
Read accesses to location 3FEh and 7FEh do not require use of the MINTACC access byte (due to its single byte status). All write accesses to 3FEh must use the MINTACC byte, however. (See *Handling Mx4 cnC++ Software/Hardware Interrupts*)				
HINTACC	3FDh	RO	RW	A host access flag byte. When the host needs to access location 3FEh, 7FEh or the status registers window (000h - 08Dh), it sets this byte equal to 01h. Mx4 cnC++ must test this flag to see if values can be written to or read from the window.
HOSTINT1	3FEh	RW	RW	HOSTINT1 is a duplicate of HOSTINT2 (7FEh). Host hardware interrupt setting and resetting is done via HOSTINT2 (7FEh). bit 0: interrupt source is buffer breakpoint bit 1: interrupt related to DSPSTAT1 register bit 2: ESTOP is detected bit 3: vector change buffer is overflown bit 4: reset finished bit 5: data run out in ring buffer (abs[vel]>0) bit 6: interrupt related to DSPSTAT2 register bit 7: unused
reserved	3FFh	-	-	reserved

Table 12: Dual Port RAM Interrupt Registers (continued on next page)

HOSTINT2	7FEh	RW	RW	When Mx4 cnC++ sets a bit(s) in this location, a host `hardware' interrupt will be generated. The interrupt remains in force until the host accesses this location. bit 0: interrupt source is buffer breakpoint bit 1: interrupt related to DSPSTAT1 register bit 2: ESTOP is detected bit 3: vector change buffer is overflown bit 4: reset finished bit 5: data run out in ring buffer (abs[vel]>0) bit 6: interrupt related to DSPSTAT2 register bit 7: unused
DSPINT	7FFh	RW	WO	When the lost sets a bit(s) in this location, a Mx4 cnC++ interrupt will be generated. The interrupt remains in force until Mx4 cnC++ accesses this location. This register currently not used.

Table 12 Cont.: Dual Port RAM Interrupt Registers

Cubic Spline Contouring Ring Buffer

(locations 400h -7F1h)

NAME	ADDRESS	ACCESS		DESCRIPTION
		DSP	HOST	
RINGBUF	400h - 7EF	RO	WO	672 byte ring buffer which contains host cubic spline contouring commands (4 bytes position, 4 bytes velocity) to be processed by the DSP. The length of all messages deposited by the host in this buffer must be a multiple of 8 bytes.
INPTR	7F0	RO	RW	Pointer to the next free location in INGBUF that the host will write to (expressed as an offset from the start of the buffer in multiples of 8 bytes).
OUTPTR	7F1	Rw	RO	Pointer to the next location in INGBUF that Mx4 cnC++ will read from (expressed as an offset from the start of the buffer in multiples of 8 bytes).

Table 13: Dual Port RAM Cubic Spline Contouring Ring Buffer

Communication Protocols Revisited

As is evident in *Mx4 cnC++ Dual Port RAM Organization*, many "windows" in the DPR are protected with access bytes. Table 6-11 lists each of the protected windows and its corresponding access bytes:

WINDOW	WINDOW DESCRIPTION	Mx4 ACCESS BYTE	HOST ACCESS BYTE
000h - 08Dh, 3FEh, 7FEh	Interrupt and Status Registers	3FCh	3FDh
0D3h - 0E2h	Actual Position	0C3h	0CBh
0E3h - 0F2h	Actual Velocity	0C4h	0CCh
0F3h - 102h	Following Error	0C5h	0CDh
103h - 112h	Index Position	0C6h	0CEh
113h - 114h	Encoder/Servo Status	0C7h	0CFh
0A7h - 0B6h	Probe Position	0C8h	0D0h
097h - 0A6h	Multi-Turn Position	0C9h	0D1h

Table 6-11: Access Bytes for DPR Windows

(It is noteworthy to remember that single byte values in the DPR are always protected by the arbitration hardware built into the DPR.)

A typical protocol that a host would use to access a 'protected' window would be:

1. Host writes 01h to the host access byte
2. Host polls the DSP access byte until it reads 00h
3. Host accesses window
4. When the host is finished with the window, host writes 00h to the host access byte.

Following this convention when accessing the windows listed in the above table ensures data integrity.

The RTC Window of the DPR is not protected with the access byte scheme, however it does use a different access protocol. As will become evident in later sections, Mx4 cnC++ checks for RTCs by looking for a host-written command code in location 3C2h. If Mx4 detects a command code, it interrupts the RTC data

and when finished, writes a zero to the RTC command code register (3C2h). Therefore, the host knows it can send an RTC command code only when location 3C2h has a zero value and Mx4 knows there is an RTC to process when location 3C2h contains a non-zero command code. The host should follow a procedure when writing RTCs to the DPR such as:

1. Host polls location 3C2h until it reads 00h
2. Host writes RTC data to locations 3C3h + as needed
3. Host writes RTC command code to location 3C2h

Handling Mx4 cnC++ Software / Hardware Interrupts

Mx4 cnC++ signals interrupts to the host computer through both hardware and software. The host has the option of responding to "hardware" interrupts across the bus via an interrupt source routine or simply polling for "software" interrupts in the Mx4 cnC++ DPR.

Mx4 cnC++ signals interrupts to the host by setting a bit(s) in the DPR's 7FEh location. This in turn, generates a hardware interrupt to the host via the bus interrupt signals. The interrupt type and interrupt source information from Mx4 cnC++ is written to the Status Registers Block of the DPR via the DSPSTAT1, DSPSTAT2 and INTAXIS registers.

The host may check for interrupts by "software" by polling location 7FEh for set bits. An important exception to the communication protocols of the previous section is made here: read accesses (or polling) to location 7FEh do not require the use of the access bytes due to its single byte status. All host-write accesses to 7FEh must, however, use the MINTACC access byte. If the host detects an interrupt by polling 7FEh, it may interrogate the proper status registers (remember to use the access bytes) for interrupt type and source information.

If the host incorporates an interrupt source routine responding to bus interrupt signals, the access to 7FEh serves two purposes. First, an access to the 7FEh location terminates the hardware interrupt. Second, the 7FEh byte bit codes some interrupt source data which directs the host as to which status registers should be interrogated for further interrupt type and source information. For example, if an axis z motion complete interrupt occurs, bit 1 of location 7FEh (HOSTINT) is set. The interrupt type is coded with a set bit 4 of DSPSTAT1 (000h). The source, axis 2, is evident as bit 1 of location 005h is set.

Note: It is important to remember that Mx4 cnC++ does not reset interrupt status register bits or 7FEh location bits. The host, after reading or recognizing an 'interrupt' location must reset bits of its own discretion.

Mx4 cnC++ Host Programming ... RTCs & Contouring

Mx4 cnC++ programming includes both contouring and RTC Mx4 cnC++ modes of motion. Typical programming applications consist of a combination of contouring and RTCs, and any combination of the two types of commands is possible of the four axes.

Real-Time Commands

Real-Time Commands (RTCs) are sent to Mx4 cnC++ via the Real-Time Command Window in the DPR (locations 3C3h to 3FBh). RTCs consist of a single byte command code and an argument list. As was introduced in *Communication Protocols Revisited*, the host should follow this procedure when writing RTCs to the DPR:

1. **Host polls location 3C2h until it reads 00h ***
2. **Host writes RTC arguments to locations 3C3h + (as needed)**
3. **Host writes RTC command code to location 3C2h (RTC command code register)**

Note: Mx4 cnC++ polls for RTCs by checking location 3C2h for a valid command code. If a valid code is detected, Mx4 cnC++ interprets the RTC and writes 00h to location 3C2h. The host should verify that Mx4 has executed a previously transmitted RTC before writing another by checking the RTC command code register for value 00h.

When writing an RTC argument list to the RTC Window, the host must follow these rules:

1. RTC argument list always starts at DPR location 3C3h.
2. RTC arguments must be written to the DPR in the order they appear as arguments in the "ARGUMENTS" declaration of RTC listing (see *Mx4 cnC++ Host-Based Programming Command Listing* in Chapter 5).
3. When writing multi-byte value RTC arguments, write LSB to MSB order.
4. Argument lists for multi-axis RTCs must be written to the DPR in increasing-axis-number order.

These rules are illustrated in the following examples that depict RTCs written to the Mx4 cnC++ DPR Real-Time Command Window.

Note: DPR locations marked "xxh" or "not necessary" in the following examples do not need to be written to (for the examples in question). Mx4 cnC++ determines which locations contain valid data via the command code and n argument (if any).

Example 1

Preset axis 3 position to 00112233h.

Use the HOME RTC,

n : 04h
$pset_3$: 00112233h

The host must write to the following DPR locations as specified:

DPR ADDRESS	BYTE	SYMBOL	DESCRIPTION
03C2h	68h	-	RTC command code
03C3h	04h	n	single byte form
03C4h	33h	$pset_3$	low byte of low word
03C5h	22h	$pset_3$	high byte of low word
03C6h	11h	$pset_3$	low byte of high word
03C7h	00h	$pset_3$	high byte of high word
03C8h	xxh	-	not necessary
:	xxh	-	not necessary
03FBh	xxh	-	not necessary

Example 2

Assuming current positions of zero for axes 2 and 4, we want to move axis 2 to the target position of 234567h and axis 4 to the target position of 112233h. Let's also assume that we want this move to be accomplished with the slew rate velocity of 200000h (200000h/2^{16} counts/200 µsec) and acceleration of 150h (150h/2^{15} counts/(200 µsec)^2) for both axes. The values for the data parameters are:

Use the AXMOVE RTC,

n	:	0Ah
acc_2	:	0150h
pos_2	:	00234567h
vel_2	:	00200000h
acc_4	:	0150h
pos_4	:	00112233h
vel_4	:	00200000h

The host must write to the following DPR locations as specified:

DPR ADDRESS	BYTE	SYMBOL	DESCRIPTION
03C2h	60h	-	RTC command code
03C3h	0Ah	n	single byte for n
03C4h	50h	acc_2	low byte
03C5h	01h	acc_2	high byte
03C6h	67h	pos_2	low byte of low word
03C7h	45h	pos_2	high byte of low word
03C8h	23h	pos_2	low byte of high word
03C9h	00h	pos_2	high byte of high word
03CAh	00h	vel_2	low byte of low word
03CBh	00h	vel_2	high byte of low word
03CCh	20h	vel_2	low byte of high word
03CDh	00h	vel_2	high byte of high word
03CEh	50h	acc_4	low byte
03CFh	01h	acc_4	high byte
03D0h	33h	pos_4	low byte of low word
03D1h	22h	pos_4	high byte of low word
03D2h	11h	pos_4	low byte of high word
03D3h	00h	pos_4	high byte of high word

03D4h	00h	vel_4	low byte of low word
03D5h	00h	vel_4	high byte of low word
03D6h	20h	vel_4	low byte of high word
03D7h	00h	vel_4	high byte of high word
03D8h	xxh	-	not necessary
:	xxh	-	not necessary
03FBh	xxh	-	not necessary

Example 3

Set a following error interrupt at 100, 101, 102 and 103 counts for axes 1 through 4, respectively.

Use the FERINT RTC,

n : 0Fh
fer_1 : 0064h
fer_2 : 0065h
fer_3 : 0066h
fer_4 : 0067h

The host must write to the following DPR locations as specified:

DPR ADDRESS	BYTE	SYMBOL	DESCRIPTION
03C2h	67h	-	RTC command code
03C3h	0Fh	n	single byte for n
03C4h	64h	fer_1	low byte
03C5h	00h	fer_1	high byte
03C6h	65h	fer_2	low byte
03C7h	00h	fer_2	high byte
03C8h	66h	fer_3	low byte
03C9h	00h	fer_3	high byte
03CAh	67h	fer_4	low byte
03CBh	00h	fer_4	high byte
03CCh	xxh	-	not necessary
:	xxh	-	not necessary
03FBh	xxh	-	not necessary

Contouring

Contouring commands consist of segment move commands transferred from the host to Mx4 cnC++ via the Contouring Ring Buffers [2nd order; DPR locations 120h to 3bfh, cubic spline; DPR locations 400h to 7EFh]. Each segment move consists of a 32-bit position value and 32-bit velocity value for each axis included in the contouring motion. The ring buffer size is (2nd order; 672 bytes) (cubic spline; 1008 bytes), and thus will hold (2nd order; 84) (cubic spline; 126) segment move [position, velocity] commands'.

Mx4 cnC++ performs either 2nd order or cubic spline interpolation on the 8-byte segment move data points. The interpolation time interval is programmable via the BTRATE (2nd order) or CUBIC_RATE (cubic spline) commands. The segment move 'commands' are executed in sequence, with execution commencing only when the previously commanded segment move is complete.

The following contouring discussion is specific to 2nd order contouring. With reference to the cubic spline contouring ring buffer, the discussion is relevant to cubic spline contouring.

Before beginning a contour, the 2nd order contouring ring buffer must first be initialized with contouring points (segment move commands). The host must load the segment move commands into the DPR in round-robin format. The following rules must be followed when initializing the ring buffer with data.

1. Data should begin in the segment command data area indicated by the value of the Mx4 cnC++ pointer OUTPTR, loaded at incrementing addresses.

2. Position and velocity are interleaved, position first.

3. Multi-byte position and velocity are written to the ring buffer in LSB to MSB format.

4. For multi-axis contouring, the position/velocity pairs for each axis involved are interleaved, written to the ring buffer in increasing-axis-number order.

5. Host should update the value of the host pointer INPTR to [offset of last segment move command + 1].

Note: The ring buffer is structured such that the next byte after location 3BFh is at location 120h; so the host must implement a roll-over to 120h after location 3BFh when loading data to the ring buffer (and/or a roll-over from 83 to 0 in the value of host pointer INPTR).

These rules are illustrated in the two examples of Fig. 6-4.

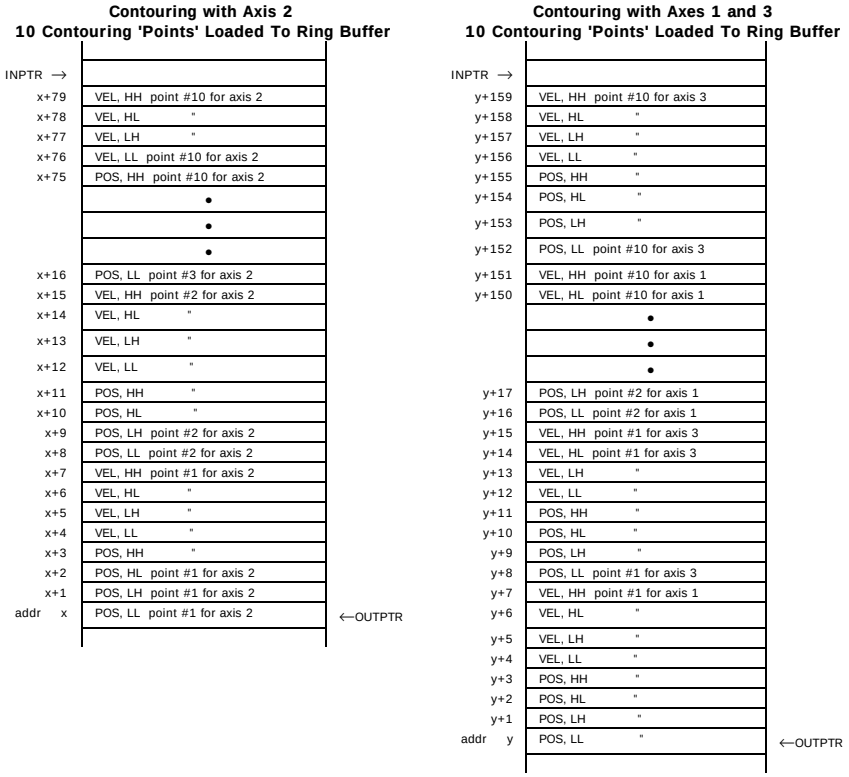


Fig. 6-4: Contouring Ring Buffer Example

As is evident in Rules 1 and 2, the DPR ring buffer area includes two pointers, INPTR and OUTPTR. Both pointers have values expressed as an offset from 120h in multiples of 8 bytes (or a single segment move command). The pointer values range from 0 to 83, pointing to one of the 84 segment move command data areas in the ring buffer.

OUTPTR is a Mx4 cnC++ pointer. It indicates which of the 84 data areas of the ring buffer Mx4 cnC++ will read the next segment move command from. Mx4 cnC++ increments OUTPTR as it reads data from the ring buffer. INPTR is a host pointer. Its value indicates which data in the ring buffer the host should begin to download additional contour data points.

Before initializing the ring buffer with contour data points, INPTR should equal OUTPTR indicating that the ring buffer is empty. The host increments INPTR as the ring buffer is initialized with data. After ring buffer initialization it is essential to ensure that INPTR always leads OUTPTR so that Mx4 cnC++ is never starved of segment move commands to read after a START RTC is issued.

The host may issue a buffer breakpoint interrupt command (BBINT RTC) in order that Mx4 cnC++ interrupts the host whenever the number of segment move commands in the ring buffer falls below a programmed threshold. This provides a system of informing the host when it should refresh the ring buffer with additional segment move commands. The number of segment move commands indicated by BBINT must always be greater than the number of segment move commands read by Mx4 cnC++ during a ring buffer refresh to prevent starvation.

When refreshing the ring buffer with additional segment move commands, Rules 2 through 5 should still be followed. Rule 1 is altered as follows:

1. Data should begin in the segment move command data area indicated by value of host pointer, INPTR.

The START RTC starts the contouring motion with the argument of START being a bit coding of the axes involved. Fig. 6-5 depicts a flowchart for a general host contouring algorithm.

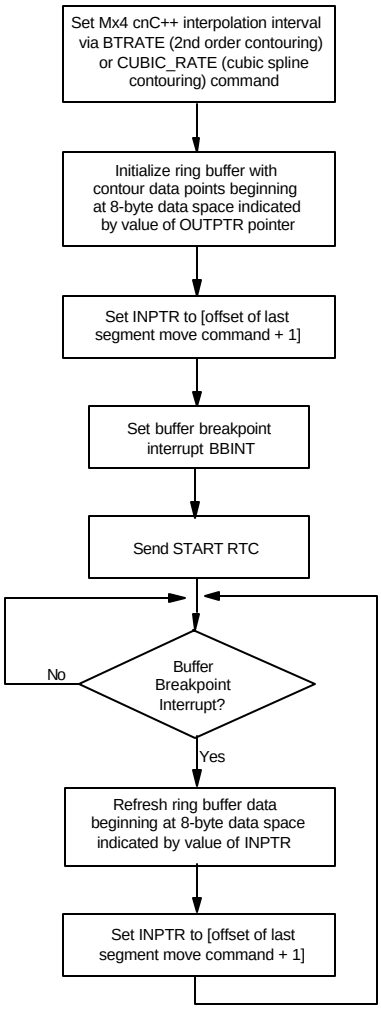


Fig. 6-5: Ring Buffer Contouring Algorithm Flowchart

Mx4 cnC++ allows the axes involved in a contouring motion to be changed "on the fly" via the VECCHG (vector change) command (2nd order) or velocity argument bit coding (cubic spline). For 2nd order contouring, upon the execution of VECCHG, the contouring task assumes a new set of axes at the programmed segment move command data space in the ring buffer. The VECCHG is triple buffered in Mx4 cnC++, so up to three vector changes may be queued at any

time. Mx4 cnC++ sends a "vector change buffer overflown" interrupt to the host if the host attempts to queue more than three vector changes.

Changing contouring axes with cubic spline requires only a change in the axis-coding bits in the velocity argument upper nibble (see *Cubic Spline Application Notes*).

Contouring motion in a particular axis may be terminated with a STOP command, or if the emergency stop ESTOP_ACC input is active. Attempting to execute a closed loop motion command such as VELMODE or AXMOVE for an axis while that axis is involved in contouring motion (or vice versa) will result in a "conflicting commands detected" host interrupt and the second command will be ignored.

Mx4 cnC++ Host Programming Using C, C++, Visual Basic or Visual C++

Programming for the Mx4 cnC++ card with the Host programming method may involve many of the following programming items:

- transmitting RTCs to Mx4 cnC++
- sending contouring commands to Mx4 cnC++ in contouring applications
- create an interrupt service routine to process any Mx4 cnC++-to-host interrupts
- check Mx4 cnC++ status bytes and error codes
- read Mx4 cnC++ system state variables such as position, velocity and following error for user-feedback
- utilize the PARREAD RTC for debug support

Mx4 cnC++'s powerful instruction set and comprehensive DPR data reporting format enables the user to create application programs from the very simple to complex. The experienced programmer may want to write low-level code that deals with Mx4 cnC++ at the bit level through the 2K DPR interface. Others, however, might want to start out with higher-level Mx4 cnC++ programming; utilizing pre-defined functions and routines that take care of the lower-level Mx4 cnC++ programming aspects such as reading and writing bytes and utilizing the correct Mx4 cnC++ DPR communication protocols.

For further information about programming the Mx4 cnC++ with C, please refer to the *Mx4 & C Programmer's Guide*.

For further information about programming the Mx4 cnC++ with C++, Visual Basic or Visual C++, contact DSP Control Group.

Mx4 cnC++ Power-Up / Reset Software Initialization

A typical Mx4 cnC++ host programming application program should include a standard initialization routine. Upon power-up, the Mx4 cnC++ card resets itself just as it would as if it had received the RESET command from the host. After the Mx4 cnC++ reset sequence is completed (signaled by 'reset complete' interrupt to the host), the Mx4 cnC++ (and VECTOR, if installed) signature is written to the DPR. If the post-reset signature is not complete, the reset was unsuccessful and the Mx4 cnC++ system reset should be repeated. Following this initialization sequence ensures that Mx4 cnC++ (and VECTOR4, if installed) has been reset and is ready to be programmed before the host programming commences. The suggested initialization sequence is depicted in the flowchart of Fig. 6-6.

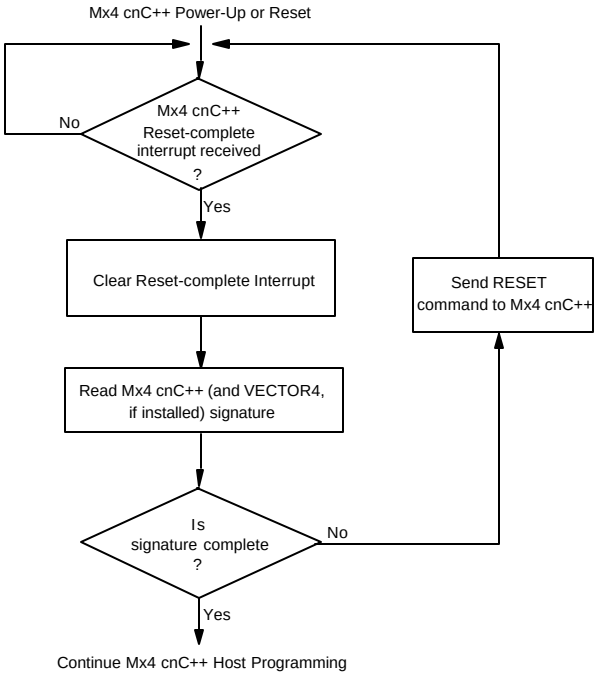


Fig. 6-6: Mx4 cnC++ Power-Up / Reset Software Initialization Routine

7 Mx4 cnC++ Status & Error Reports

Mx4 cnC++ Power-Up / Reset State

Upon power-up, the Mx4 cnC++ card resets itself just as it would as if it had received the RESET RTC from the host. Upon completion of the reset, Mx4 cnC++ sends the 'reset finished' interrupt to the host. Thus, upon power-up of the Mx4 cnC++ card, the host should wait for and then clear the 'reset finished' interrupt generated by Mx4 cnC++. (See Chapter 6, *Mx4 cnC++ Power-Up / Reset Software Initialization*.)

A Mx4 cnC++ reset results in the clearing of all Mx4 cnC++ parameters including the 2K Dual Port RAM (DPR). Any programming of the Mx4 cnC++ card prior to reset must be repeated.

Mx4 cnC++ Interrupts, Status Codes & Error Condition Reports to the Dual Port RAM

Mx4 cnC++ data reporting to the host includes a variety of interrupt conditions, status codes and error conditions. Table 7-1 sums up these reports.

Note: Refer to Chapter 6 *Mx4 cnC++ Host-Based Programming* for a detailed description of the Mx4 cnC++ DPR interface and protocols for host writing and reading of the DPR.

NAME	HOST INTERRUPT	RTC ENABLED	DESCRIPTION
Following Error and Halt Interrupt	√	√	see FERHLT command description
Following Error Interrupt	√	√	see FERINT command description
Index Pulse Interrupt	√	√	see INXINT command description
Position Breakpoint Interrupt	√	√	see POSBRK command description
Motion Complete Interrupt	√	√	see MCENBL command description
External Interrupt	√	√	see PRBINT command description
Conflicting Commands Detected	√		"Conflicting commands detected" is an error interrupt reported to the host via bit 6 of DPR status register DSPSTAT1 (see *Mx4 DPR Organization*). This interrupt serves to report the host error of attempting to combine both contouring and RTC motion in the same axis simultaneously. Any one of the following cases will yield a "conflicting commands detected" interrupt. For an axis with 1. contouring in progress, and an AXMOVE or VELMODE RTC involving that axis is sent to Mx4 2. AXMOVE or VELMODE motion in progress, and a contouring START RTC involving that axis is sent to Mx4. The "conflicting command(s)" that caused the error is not executed by Mx4, it is discarded.

Table 7-1: Mx4 Interrupts, Status Codes and Error Conditions (continued on next page)

NAME	HOST INTERRUPT	RTC ENABLED	DESCRIPTION
RTC Command Ignored	√		This error interrupt is reported to the host via bit 7 of DPR status register DSPSTAT1 (see *Mx4 DPR Organization*). The "RTC command ignored" interrupt is generated when a motion command is received by Mx4 for an axis that is presently halting to a stop via a previously executed STOP RTC or active ESTOP/ input. The motion commands are the AXMOVE, START or VELMODE RTCs.
Positive Feedback Interrupt	√	√	see POSFEED command description
Encoder Lost Interrupt	√	√	see ENCOLOS command description
Offset Cancel Finished	√	√	see OFFSET command description
Encoder Status			The "encoder status" is 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 following error count is > 300h, and 2. 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.
Real Time Index Pulse Reporting			In addition to the host-enabled index pulse interrupt, the host may monitor the index pulses of all four axes via the "Real time index pulse reporting" in the upper nibble of DPR location 113h (see *Mx4 DPR Organization*). A set bit indicates an index pulse and the nibble is updated in real-time (5 msec).

Table 7-1 cont.: Mx4 Interrupts, Status Codes and Error Conditions (continued on next page)

NAME	HOST INTERRUPT	RTC ENABLED	DESCRIPTION
Servo Check Status			The "servocheck status" byte (SERVOCHK, DPR location 114h) is used to report a Mx4 internal stack overflow error. Upon detection of a set bit 0 of SERVOCHK, the host may need to perform a system reset.
Buffer Breakpoint Interrupt	√	√	see BBINT command description
ESTOP Detected	√		Mx4 reports the occurrence of an emergency stop (ESTOP/ input) to the host with an interrupt. The interrupt is reported via bit 2 of DPR interrupt register HOSTINT (see *Mx4 DPR Organization*). When Mx4 detects an active ESTOP/ input, the motion (if any) of all four axes is brought to a halt with the programmed ESTOP/ acceleration / deceleration.
Vector Change Buffer Overflown	√		The VECCHG RTC utilizes three buffer levels to implement the instruction. If the host attempts to "stack" or buffer more than three VECCHG commands, a host interrupt is generated and reported via bit 3 of DPR interrupt register HOSTINT (see *Mx4 DPR Organization*).
Reset Finished	√		Upon completion of the power-up sequence or RESET RTC, Mx4 signals an interrupt to the host. The interrupt is bit-coded as a set bit 4 of DPR interrupt register HOSTINT (see *Mx4 DPR Organization*).
Data Run-Out In Ring Buffer	√		It is essential for the host to provide Mx4 segment move commands via the ring buffer(s) while contouring is in progress. If at any time while contouring is active Mx4 detects INPTR=OUTPTR and any activated axis velocity is not zero, the ring buffer is out of data and Mx4 sends a "data run-out in ring buffer" interrupt to the host. The interrupt is recorded in bits of DPR interrupt register HOSTINT (see *Mx4 DPR Organization*).

Table 7-1 cont.: Mx4 Interrupts, Status Codes and Error Conditions

8 VECTOR4

Note: The following is a brief description of the functionality and capabilities of the VECTOR4 drive control option. A more detailed VECTOR4 description can be found in the *VECTOR4 User's Guide* (included with the VECTOR4 option).

VECTOR4 is an add-on daughter-board to the Mx4 cnC++ card (PC/AT, Multibus and VME). VECTOR4 is an all digital AC servo controller for four axes of both AC and DC motors. With the VECTOR4 add-on drive controller, Mx4 cnC++ controls brushless DC, variable reluctance, AC induction and brush-type DC motors. Any combination of these motor technologies is programmable through a single instruction. Some of the features of VECTOR4 are listed below:

- Closes current and velocity loops for four axes
- State feedback optimum control algorithm along with modified vector control for better stability
- Programmable power electronics parameters
- PWM (Pulse Width Modulation) outputs allow VECTOR4 to interface to any generic output stage
- Current limit is programmable
- Full VECTOR4 debug support and state variable parameter reporting
- Performs field weakening for brushless DC and AC induction motors
- Current loop offset adjustment
- Interfaces to Mx4 cnC++ through the DSP bus

The VECTOR4 card interfaces to Mx4 cnC++ via two 25-pin headers (see Fig. 8-1). The Mx4 cnC++/VECTOR4 combination requires a double-slot width in the host bus card cage.

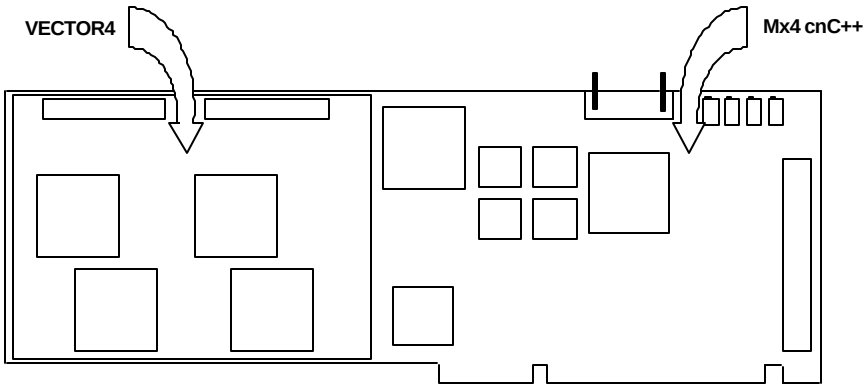


Fig. 8-1: PC/AT Mx4 cnC++ with VECTOR4 Option

Mx4 cnC++ transmits to VECTOR4 a set of torque or velocity commands through the DSP bus. VECTOR4 closes the velocity and/or torque loops and performs commutation (see VECTOR4 functional block diagram of Fig. 8-2). VECTOR4 uses state variable control rather than classical techniques such as lead-lag or PID. In addition to providing superior performance, this approach simplifies the task of tuning system parameters during initial system set-up. VECTOR4 linearizes the torque and current relations and closes a single composite current loop similar to that of a DC brush-type motor. This technique is superior to conventional vector control which requires closing three current loops and results in a motor-speed dependent current loop frequency response.

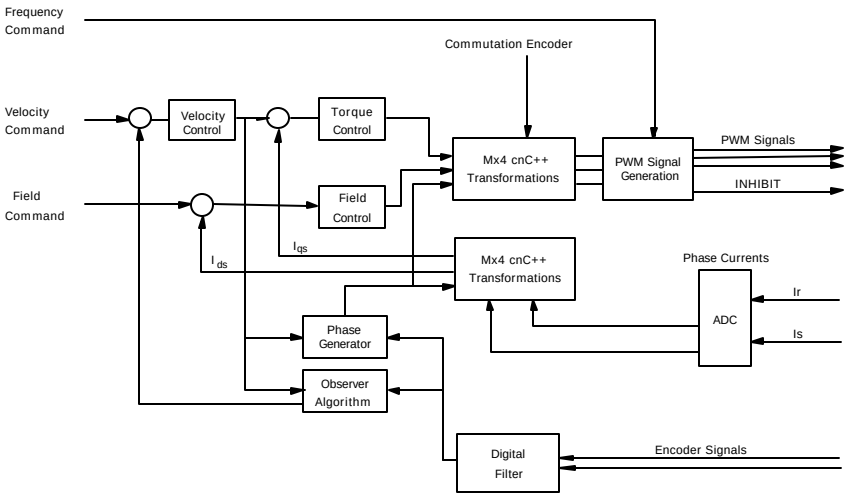


Fig. 8-2: Mx4 cnC++/VECTOR4 Functional Block Diagram

An industrial motor may be a brush type DC, brushless DC or AC induction machine. Instructions provided in VECTOR4 enable the user to select the appropriate technology suitable for their application. The encoder mounted on the motor shaft may be an incremental encoder alone (for AC induction motor or brush type DC motor control) or the combination of incremental and commutation encoders (for brushless DC motor control applications).

The commutation techniques used for brushless DC technology use only a 3-bit hall sensor or similar, as well as incremental encoder signals. This is in contrast with the use of an expensive resolver and resolver-to-digital converter. The incremental encoder signals enter into the Mx4 cnC++ card through the card connectors, whereas commutation encoder signals interface directly to VECTOR4.

VECTOR4 Programming Capabilities

Like Mx4 cnC++, VECTOR4 is based on a powerful kernel that provides a very flexible programming platform. VECTOR4 supports both the host based and DSPL programming methods. VECTOR4's commands provide the user an easy-to-use interface to VECTOR4's field-oriented control core. The VECTOR4 command summary is categorized as follows:

Initialization

Initialization commands encompass those instructions used to define a particular set-up. For example, programming the type of motor being used or the encoder characteristics of an axis.

COMMAND	DESCRIPTION
ENCOD_MAG	encoder line number
FLUX_CURRENT	bipolar field flux value
MOTOR_PAR	set motor parameter
MOTOR_TECH	select motor technology
VECTOR4_BLOCK	block further instructions to VECTOR4

Control Parameter

Instructions used to set state variable control parameters and to tune the control loops.

COMMAND	DESCRIPTION
CURR_OFFSET	current loop offset adjustment
CURR_PID	program current loop gains

Power Stage

Power stage commands are used to characterize and define the VECTOR4-output stage interface.

COMMAND	DESCRIPTION
CURR_LIMIT	current limit setting
PWM_FREQ	set VECTOR4 PWM output frequency

System Diagnostic

System diagnostic instructions provide the Mx4 cnC++ programmer powerful debug support.

COMMAND	DESCRIPTION
VIEWVEC	select VECTOR4 variable feedback

This page intentionally blank.

9 Mx4 cnC++ Specifications

Performance

ITEM	DESCRIPTION
Servo Loop Update	120 μsec (all axes included)
Control Algorithms	State Feedback Multi-Input Multi-Output Controller, Kalman Filter, Robust Control, PID, Notch
Block Execution Rate (programmable)	2nd Order: 5 to 20 msec Cubic Spline: 1 to 100 msec
Position Range	+/- 2, 147, 483, 650 counts, rollover, +/- 1 count
Position Capture	100 ns max. delay from trigger
Velocity Range	0 to 1,280,000 counts/sec, +/- 1 count
Acceleration Range	0 to 50,000,000 counts/sec^2, +/- 1 count
Synchronization	Unlimited number of axes can be synchronized to the same servo cycle

Hardware

ITEM	DESCRIPTION
Processing	Quad DSPs in Hyper-Cube Architecture, 36 MIPS
Analog Output	16-bit parallel DAC per axis, 300 uV resolution, -10v to +10v output

Input / Output

ITEM	DESCRIPTION
Protective Inputs	ESTOP/ emergency stop [1]
External Interrupts	/PR0, /PR1 [2]
General Purpose Inputs	User-defined inputs [10], TTL logic
General Purpose Outputs	User-defined outputs [3], TTL logic

Position Encoder Feedback

ITEM	DESCRIPTION
Encoder Type	A/B quadrature or single-ended with "I" (index pulse) channel [optional]
Maximum Encoder Count	5 MHz
Maximum Encoder Pulse	1.25 MHz

Electrical

ITEM	DESCRIPTION
+5v Mx4 cnC++ Supply Voltage	MIN = 4.75v NOM = 5v MAX = 5.25v
Operating Free Air Temperature Range	0°C to 70°C

Power Consumption

ITEM	DESCRIPTION
+5v	MAX = 3A
+12v	MAX = 250mA
-12v	MAX = 250mA

Mechanical

ITEM	DESCRIPTION
PC/AT Mx4 cnC++ Interface Connector	protected header, .100 x .100 centers, center and dual polarized
PC/AT Mx4 cnC++ I/O Connector	protected header, low profile, .100 x .100 centers, center polarized
PC/AT Mx4 cnC++ Synch Connector	AMP 640457-4 friction lock header
Mechanical Dimensions	See Fig. 9-1

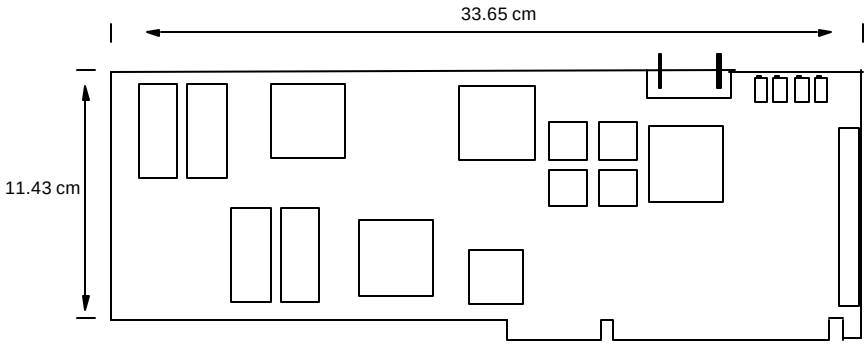


Fig. 9-1: PC/AT Mx4 cnC++ Mechanical Dimensions

This page intentionally blank.