CHAPTER ④

Application Design

Chapter Objectives

The information in this chapter will enable you to:

- ❑ Recognize and understand important considerations that must be addressed before you implement your application
- ❑ Understand the capabilities of the system
- ❑ Use examples to help you develop your application

Motion Profile Application Considerations

This section contains information that you should consider and evaluate when designing and developing your system.

Positional Accuracy vs. Repeatability

Some applications require high absolute accuracy. Others require repeatability. You should clearly define and distinguish these two concepts when you address the issue of system performance.

If the positioning system is taken to a fixed place and the coordinates of that point are recorded. The only concern is how well the system repeats when you command it to go back to the same point. For many systems, what is meant by accuracy is really repeatability. Repeatability measures how accurately you can repeat moves to the same position.

Accuracy, on the other hand, is the error in finding a random position. For example, suppose the job is to measure the size of an object. The size of the object is determined by moving the positioning system to a point on the object and using the move distance required to get there as the measurement value. In this situation, basic system accuracy is important. The system accuracy must be better than the tolerance on the measurement that is desired. Consult the technical data section of the Compumotor Catalog for more information on accuracy and repeatability.

Move Times—Calculated vs Actual

You can calculate the time it takes to complete a move by using the acceleration, velocity, and distance values that you define. However, you should not assume that this value is the actual move time. There is calculation delay and motor settling time that makes your move longer. You should also expect some time for the motor to settle into position. The SX has minimal calculation-delay time associated with a Go (**G**) command. This delay can be as low as 500 µs. The SX has an internal timer that allows you to monitor the elapsed time of your move. The response of the **TM** command shows you the previous move's execution time.

Preset Mode Moves

A preset move is one in which the distance is specified in motor steps. You can select preset moves by putting the SX into Normal mode with the Mode Normal (**MN**) command. Preset moves allow you to position the motor in relation to the motor's previous stopped position (incremental moves) or in relation to a defined zero reference position (absolute moves). You can select incremental moves with the Mode Position Incremental (**MPI**) command. You can select absolute moves with the Mode Position Absolute (**MPA**) command. At any time, you can request the state in which the SX is configured by issuing the **DR** command.

Incremental Mode Preset Moves

When you are in Incremental mode (**MPI**) and Preset mode (**MN**), the motor moves the specified distance from its current position. The preset distance is specified with the **D** command. You can specify the direction with the optional sign (**D+8000** or **D-8000**), or you can define it separately with the Change Direction (**H+** or **H-**) command. If no sign is specified with the **D** command it defaults to positive.

Command	Description
> LD3	Disables CW and CCW limits (**not needed if limits are installed**)
> MPI	Sets unit to Incremental Position Mode
> MN	Places the SX in Preset mode
> PZ	Zeroes the position counter
> A25	Sets acceleration to 25 rps^2
> AD25	Sets deceleration to 25 rps^2
> V5	Sets velocity to 5 rps
> D8000	Sets distance to 8,000 steps
> G	Executes the move (Go)
> D12000	Sets distance to 12,000 steps
> G	Executes the move (Go)
> 1PR	Reports the setpoint (commanded) position Response: *+**00000020000**

Absolute Mode Preset Moves

A preset move in the Absolute mode (**MPA**) and Preset mode (**MN**) moves the motor to the distance in an absolute coordinate system that you specify relative to an absolute zero position. You can set the absolute position to zero with the Position Zero (**PZ**) command or by cycling the power to the Indexer. The absolute zero position is the initial power-up position.

The direction of an absolute preset move depends upon the motor position at the beginning of the move and the position you command it to move to. If the motor is at absolute position +12,800, and you instruct the motor to move to position +5,000, the motor will move 7,800 steps in the negative direction to reach the absolute position of +5,000.

The SX powers up in Incremental mode. When you issue the Mode Position Absolute (**MPA**) command, it sets the mode to absolute. When you issue the Mode Position Incremental (**MPI**) command the unit switches to Incremental mode. The SX retains the absolute position, even while the unit is in the Incremental mode. You can use the Position Report (**PR**) command to read the absolute position.

In the following example, the motor performs the same commands as the incremental position example. In this case, the **PR** command will report a different position because it is working in an absolute coordinate system.

Command	Description
> LD3	Disables CW and CCW limits (**not necessary if limits are installed**)
> MPA	Sets unit to Absolute Position mode
> PZ	Zeroes the position counter
> A25	Sets acceleration to 25 rps^2
> AD25	Sets deceleration to 25 rps^2
> V5	Sets velocity to 5 rps
> D8ØØØ	Sets distance to 8,000 steps
> G	Executes the move (Go)
> D12ØØØ	Sets distance to 12,000 steps
> G	Initiates motion
> 1PR	Reports the setpoint (commanded) position Response: *+**ØØØØØ12ØØØ**

The motor will move to absolute position 8,000. The second move is 4,000 more steps to the absolute position of 12,000 steps. The **PR** command reports a setpoint (commanded position) of 12,000 steps.

Continuous Mode Moves

The Continuous mode (**MC**) command accelerates the motor to the velocity that you last specified with the Velocity (**V**) command. The motor continues to move at the specified velocity until you issue the Stop (**S**) or Kill (**K**) command or specify a velocity change. To interactively change velocity while the motor is moving, use the instantaneous velocity command (**IV**). To change velocity on-the-fly in a sequence, use the Motion Profiling mode (**MPP**).

In Motion Profiling mode, all buffered commands are executed immediately—therefore you only have to enter the **V** command to change the velocity. No **G** is needed following the **V**. Continuous mode is useful for applications that require constant movement of the load, and the motion is not based on distance but is based on internal variables or external inputs, or when the motor must be synchronized to external events such as trigger input signals. In this example, velocity is changed after a time delay of about 1 second.

Command	Description
> PS	Pauses motion until a **C** command is reached
MPP	Places the SX in the **MPP** mode
IN1A	Sets up **I1** (Input 1) as trigger bit 1
IN2A	Sets up **I2** (Input 2) as trigger bit 2
LD3	Disables CW and CCW limits (**not necessary if limits are installed**)
MC	Sets unit to the Continuous mode
A25	Sets acceleration to 25 rps^2
AD25	Sets deceleration to 25 rps^2
V1	Sets velocity to 1 rps
G	Executes the move (Go)
T1	Waits 1 second after the move is started
V5	Sets velocity to 5 rps
TR1Ø	Waits for trigger bit 1 to go on and bit 2 to go off
STOP	Stops the motor
C	Continues execution of commands

These commands cause the SX to run in Continuous mode. The motor starts accelerating, waits for 1 second, changes velocity to 5 rps, waits for you to turn **I1** (input 1) on and turn **I2** (input 2) off, and then stops. *The* **VØ** *and* **STOP** *commands stop the motor (the* **S** *command is not a buffered command and cannot be used in this situation, unless you wish to halt the operation in the middle of the program).* The **DIN** command (an immediate command) simulates the state you want the inputs to be in. In the example above, you could simulate the activation of the trigger state without physically toggling the inputs, by using the **DIN** command as follows.

> DINEEEE1ØEEEEEE

E means *do not affect the input.* A 1 makes the input one, a 0 makes the input zero. Each **E**,1, or 0 represents an input bit. There are 12 inputs. The 1 and Ø in this example correspond to **I1** and **I2** on the front panel. The first 4 **E'**s correspond to CCW, CW Home limit, and Registration Input.

Closed Loop Operation

As of January 1, 1995, the SX/SXF will no longer have absolute encoder interface capability as a standard feature. The standard SX/SXF will not be compatible with the AR-C absolute encoder, but rather the SX/SXF absolute encoder interface will be an option to the standard system. For help in determining whether or not your SX/SXF has the absolute encoder interface, see the RVV command in the ***SX Software Reference Guide.***

This section explains the closed-loop operation of the SX. Closed-loop moves use external sensors to provide position verification signals. Closed-loop operation provides the SX with the ability to detect a system stall or to adjust the load position to compensate for the system's mechanical slop.

The standard SX-A can interface only to an incremental encoder. The SX-A can interface to an incremental encoder or a Compumotor absolute encoder (AR-C). The encoder may be used as a means of creating a closed-loop system or as an independent means of verifying motor position. The following functions are added to a system when an encoder is used:

- ❏ Encoder referenced positioning
- ❏ Encoder position correction
- ❏ Motor stall detection
- ❏ Higher accuracy homing function

To implement the closed-loop functions, you must connect an incremental encoder or a Compumotor AR-C absolute encoder to the SX. The SX can supply up to 250mA/+5VDC to power the incremental encoder (the AR-C provides it's own power). When you use incremental encoders with single-ended outputs, do not connect channels A-,B-, and, Z- to the SX's encoder connector. Refer to *Chapter 3, Installation* for details on wiring encoders to the SX.

Selecting Encoder Resolution Values: Incremental

The number of encoder steps that the SX system recognizes is equal to four times the number of encoder lines. For example, a 1000-line encoder mounted directly on the motor will generate 4000 encoder steps per revolution of the motor shaft. The SX reads in a quadrature signal.

Selecting Encoder Resolution Values: Absolute

The number of encoder steps that the SX system recognizes is equal to the specified number of absolute encoder steps that are selected at the AR-C decoder box. The absolute encoder is capable of providing either 16384, 8192, 4096, or 2048 steps per revolution.

Motor-to-Encoder Ratios

A minimum of three motor steps per encoder step is required for successful operation of the Position Maintenance function. If a 1000-line incremental encoder is mounted directly to the motor shaft, the motor must have a resolution of 12,000 steps/rev or higher. Ratios above three motor steps per encoder step ensure stability of the position maintenance correction function. **When operating in Encoder Step mode, system accuracy depends on the accuracy of the encoder and not on the accuracy of the drive and motor combination in Open-Loop mode.**

If you install a reducer (gear box) between the motor shaft and the encoder, the number of encoder steps that the Indexer receives is equivalent to the number of encoder steps divided by the encoder gear ratio.

For example, using a 12,800 steps/rev motor, a 1,000-line encoder, and a 10:1 reducer, the ratio of motor revolutions to encoder steps would be changed as described in the following table.

Parameter	1:1 Ratio	10:1 Ratio
Number of encoder steps recognized by the SX (per motor rev)	4,000	400
Required ratio of motor revs to encoder steps	1/4,000	1/400
Motor-to-encoder step ratio	3.2/1	32/1

Setting Encoder Resolution

Incremental Encoder

Encoders are available in a wide range of resolutions. Using the Encoder Resolution (**ER**) command, you must specify the resolution of the encoder that is connected to the SX.

Absolute Encoder

Compumotor offers an absolute encoder for use with the SX-A system. The AR-C absolute encoder is capable of providing one of four resolutions (16384, 8192, 4096, or 2048 steps/rev). The encoder resolutions are selected using DIP switches inside of the AR-C decoder box.

Using the following example as a guide, you can specify the encoder's resolution to the with the Encoder Resolution (**ER**) command.

Commmand	Description
> ER4000	Sets encoder's post-quadrature resolution to 4000 steps/ rev (1,000 lines)
> ER8192	Sets absolute encoder resolution to 8192 steps/rev

The number entered with the ER command is the number of encoder steps that will be counted when the motor has moved 1 revolution. If a 1000-line encoder is directly mounted to a 25000 step motor, there will be 4000 encoder steps for 1 motor revolution.

Encoder Step Mode

The Indexer can perform moves in either motor steps or encoder steps. In Motor Step mode (**FSBØ**), the distance (**D**) command defines moves in motor steps. In Encoder Step mode (**FSB1**), the distance command defines moves in encoder steps. When the AR-C is used, the Enable Absolute Encoder command (**FSM1**) must be used prior to enabling the Encoder Step mode (**FSB1**). You must set up the Indexer for the correct encoder resolution. The Encoder Resolution (**ER**) command defines the encoder resolution. The sample move below gives examples of incremental and absolute encoder step moves.

Incremental Encoder

Command	Description
> LD3	Disables CW and CCW limits (not necessary if limits are installed)
> MN	Sets motor to Normal mode
> ER4000	Sets encoder resolution to 4000 counts/rev
> FSB1	Sets move to Encoder Step mode
> A10	Sets acceleration to 10 rps^2
> V5	Sets velocity to 5 rps
> D4000	Sets distance to 4000 encoder steps
> G	Executes the move (Go)

The motor will turn CW until 4000 encoder steps (1 rev) are received. If this move does not work, refer to *Chapter 7 , Maintenance & Troubleshooting*.

If the encoder is accidentally disconnected during the execution of this move, the motor will continue to move CW indefinitely. This is because the system continues to move, while waiting for the remaining encoder pulses. To prevent this, use the Stop-On-Stall (**FSD1**) command.

If this example causes a very slow, creeping motion, the encoder is not wired properly or the encoder resolution being seen is too low. Confirm the encoder count is increasing with CW motion using the PX command.

Absolute Encoder

Command	Description
> LD3	Disables CW and CCW limits (not necessary if limits are installed)
> MN	Sets motor to Normal mode
> MR15	Sets motor resolution to 50000 steps/rev
> FSM1	Enable absolute encoder
> 1RSE	Check for AR-C problems
> ER16384	Select (default) AR-C encoder resolution
> FSB1	Sets move to encoder step mode
> A10	Sets acceleration to 10 rps^2
> V5	Sets velocity to 5 rps
> D16384	Sets distance to 16384 encoder steps
> G	Executes the move (Go)

The motor will turn CW until 16,384 encoder steps (1 rev) are received. If this move does not work, refer to *Chapter 7, Maintenance & Troubleshooting.*

If the encoder is accidentally disconnected during the execution of this move, the motor will stop immediately as if stop-on-stall was enabled and the motor has stalled. **1RSE** will respond with ***ABSOLUTE_ENCODER_FAULT** and **1R** will respond with ***S** (attention required). To clear this fault condition, execute the **ON** command.

If this example causes a very slow, creeping motion, the encoder is not wired properly or the encoder resolution being seen is too low. Confirm the encoder count is increasing with CW motion using the PX command.

If you need to power up the SX and an absolute encoder simultaneously, enter the following commands into sequence 100.

Command	Description
> FSM0	Disable absolute encoder
> T1	Pause for 1 second
> FSM1	Enable absolute encoder

This gives the absolute encoder time to fully power-up before the SX checks the interface.

Position Maintenance

The Position Maintenance (**FSC**) command enables and disables the position maintenance function. You must enable Position Maintenance (**FSC1**) to activate closed-loop position correction and ensure that encoder step moves are positioned within the specified Deadband (**DW**) of the commanded encoder position. Enabling position maintenance will cause the Indexer to servo the motor until the correct encoder position is reached. This occurs at the end of a move (if the final position is incorrect) or any time the Indexer senses a change in position while the motor is at Ø velocity. To enable position maintenance, you must have an encoder connected and the Indexer set in Encoder Step mode (**FSB1**).

If a stall occurs during position maintenance, the position error is set equal to zero and the motor does not move. If you disable Stop-on-Stall (**FSDØ**), the SX will continue to try to move the motor the commanded distance.

☛ Note — Position Maintenance is not cancelled by a Stop (**S**) or a Kill (**K**) command. Use the **FSDØ** or **OFF** commands.

The SX's Position Maintenance mode corrects for final positional accuracy after a move is completed. If a 4,000 encoder step move is commanded, the motor will attempt to go 25,000 motor steps (or 1 rev). After the move is complete, the system checks the encoder to ensure the encoder reads 4,000 steps or 1 revolution. If it is not at 4,000, it will correct the motor's final position by commanding the motor to move until the encoder reads 4,000 steps. You can use the following commands to set up a position maintenance application.

Command	Description
DPA	Displays actual motor position indicated by the encoder—displayed in encoder steps.
PR	Displays number of steps motor was commanded to go—motor or encoder steps.
PX	Displays actual motor position indicated by the encoder—displayed in encoder steps.
DPE	**D**isplays the **P**osition **E**rror between where the motor was commanded to go (setpoint) and where the encoder indicates the motor is(actual position). In the Motor step mode, the error is displayed in motor steps. In the encoder step mode, the error is displayed in encoder steps.
CPE	**C**onfigures the maximum **P**osition **E**rror allowed. The actual motor position that is off from the setpoint or commanded position. This determines when a stall is detected (FSD1)
CPG	**C**onfigure **P**osition **G**ain. This is the percentage gain that is applied to the position error to determine a correction velocity.
CPM	Maximum gain value for which the **CPG** determines the % of to use for correction.
MV	The maximum velocity that the SX can command when correcting for position while in Position Maintenance mode.
FSC	Enters and exits Position Maintenance mode.
FSD	Enters and exits the **S**tall **D**etect mode.
DW	This is the deadband window, when the error is greater than this number of steps, the SX will perform position maintenance. A wider window Prevents dither.

In Position Maintenance mode, the SX performs an end-of-move correction. It corrects for any position error that is detected between the commanded position and the encoder position which is greater than the deadband window. The SX has a 1-ms update rate. Every ms it will correct for any position error above the deadband window. It will continue to send pulses to the motor until the motor is in position according to the encoder. Position maintenance can be used only in Encoder Step mode.

In the encoder mode, the SX determines the motor position from the encoder. A move is considered complete when the encoder position reaches the commanded position. When a step motor is accelerated or decelerated, it will ring about the commanded move trajectory. The higher the acceleration, the larger the amplitude of the ringing. If the amplitude of the ringing is larger than one motor pole, the motor and drive will lose synchronism and a stall will result. Microstepping reduces the violence of this ringing, but does not totally eliminate it.

Because of this ringing, it is possible for the motor to overshoot or undershoot its commanded goal. This occurs because at the moment the encoder indicates the correct terminal position and the SX stops commanding motion, the motor may still ring back to a different resting position. The magnitude of this final error can be anywhere from a few steps to 30-50 steps depending on the acceleration. The purpose of Position Maintenance is to correct this end of move error.

Position maintenance is also very useful in correcting for system errors if the encoder is mounted on the load rather than directly behind the motor. Mounting an encoder on the load is preferred when the system mechanics allow.

If the Position Maintenance mode (**FSC1**) is on, the motor will servo to the commanded position after the move is completed. With both Stall Detect (**FSD1**) and Position Maintenance (**FSC1**) modes activated, the SX will apply the position error to the following algorithm: If **DPE** (position error) > **CPE** (maximum allowable position error), a stall condition exists. Stall errors occur and the step and direction output is turned off. *If **DPE**(position error) > **DW** (deadband window), position maintenance occurs, unless a stall is detected*. Stalls are checked for each sample period, and position maintanance is checked each sample period at the end of a commanded move.

Correction velocity is determined by the following equation:

Velocity = **CPG** (Proportional gain percentage) • **CPM** (Proportional gain maximum) • **DPE** (position error)

If this is less than the **MV** value (maximum correction velocity), it is the commanded correction velocity. If it is greater than the **MV** value, the **MV** value is the commanded position maintenance correction velocity.

If **DPE** is < **DW**, no correction occurs. **If oscillation occurs after the move, the Position Maintenance gains may be set too high.**

Example

Perform the following steps to make a move with stall detect and position maintenance activated:

Step ①

Type the following commands:

Command	Description
> 1CPE2ØØØ	Maximum position error is 2,000 motor steps
> 1CPG5Ø	Sets position gain to 50% of maximum gain (CPM)
> 1CPM6Ø	Maximum position gain is 60
> 1MV1Ø	Sets maximum correction velocity to 10 rps
> 1DW5	Sets deadband window to 5 steps
> 1FSD1	Invokes Stall Detect mode
> 1FSB1	Set system to Encoder Step mode
> 1FSC1	Invokes Position Maintenance mode

This enables the Position Maintenance mode. The above selected parameters may make the motor unstable when unloaded.

Step ②

Type the following commands to make a move:

Command	Description
> A1ØØ	Sets the acceleration to 100 rps^2
> AD1ØØ	Sets the deceleration to 100 rps^2
> V1Ø	Sets velocity to 10 rps
> D1ØØØØØ	The motor is set to move 100,000 encodersteps
> G	Initiates motion

Step ③

Use the following commands to observe the position error and actual position.

Command	Description
> 1DPA	Display the actual position
> 1DPE	Display the position error

The motor should move to the commanded position, +100,000 encoder steps.

Stop-On-Stall

You can enable the Stop-on-Stall function with the FSD1 command. The move will terminate, without any delay, as soon as a stall is detected. This function works either in Motor Step or Encoder Step mode and is independent of position maintenance.

CAUTION

Disabling the Stop-on-Stall function with the FSDØ command will allow the SX to attempt to finish the move profile regardless of a stall detection, even if the load is jammed. This can potentially damage user equipment.

The Stop-on-Stall feature depends on the maximum allowed position error (set with the Configure Position Error [**CPE**] command). The factory default setting for maximum position error is 4000 encoder steps.

Command	Description
> CPE1ØØ	Sets maximum position error to 100 steps
> ER4ØØØ	Sets encoder resolution to 4,000 steps/rev
> FSD1	Enables Stop on Stall mode

While at rest the position error will normally never exceed 30 to 50 steps. When operating at high speeds, the difference between the command and actual position (position error) can easily exceed 50 steps. This is because the motor has already been moving for 1 ms before the position error is calculated. For example, at **MR11** (25,000 steps/rev) the position error can be 1,250 steps at 50 rps. This number could be larger by a ringing motor or motor windage.

Stall detection does not occur until the error exceeds the maximum position error (set with the **CPE** command). Consequently, if the commanded motor position and the encoder position differ by 100 motor steps, the SX will detect a stall and stop the motor immediately.

Output-On-Stall

You can define one of the outputs to act as an output-on-stall. By defining the output type as **L** it becomes an output-on-stall. Whenever a stall condition occurs the output will be activated. You can be in either Motor Step mode or Encoder Step mode, and you do not have to be in Position Maintenance mode. By selecting an output as an output-on-stall you are not causing the motor to stop on a stall. The motor will not stop on a stall unless you enable it with **FSD1**.

Command	Description
> MN	Sets the system in the Preset mode
> FSB1	Enables Encoder mode
> CPE3Ø	Selects maximum position error of 30 motor steps
> OUT1L	Enables stall detect function on output #1
> FSD1	Stops motor if a stall is detected
> A2	Sets acceleration to 2 rps^2
> V.1	Sets velocity to 0.1 rps
> D128ØØ	Sets distance to 12,800 encoder steps
> G	Execute the move (Go)

While the motor is moving, you can cause a stall by holding the shaft. If you can not manually stall the motor, you can simulate a stall by carefully disconnecting the +5V encoder lead from pin #1 on the SX encoder connector. When the stall occurs, output #1 is turned on and the motor stops (this signals you that the motor has stalled).

Fault Sequence Execution with Stall Detect

If the user has an error or kill sequence defined (**SFKn**), a stall will cause the program to execute the SFX sequence. This can be very useful in alerting machine operators to error conditions or mechanical difficulties.

Mechanical Resonance

Resonance, a characteristic of all stepper motors, can cause the motor to stall at low speeds. Most full-step motor controllers *jump* the motor to a set minimum starting speed to avoid this resonance region. This causes poor performance below 1 rps. In nearly all cases, the stepping features of the SX will overcome these problems. However, in some cases the drive will need to be optimized with some simple adjustments to overcome resonance.

Resonance occurs at speeds which approach the natural frequency of those speeds. It causes the motor to vibrate at these speeds. The speed at which fundamental resonance occurs is typically between 0.3 and 0.8 rps and is highest for small motors and lowest for large motors.

Motors that will not accelerate past 1 rps may be stalling due to resonance. The resonance point may be lowered to some extent by adding inertia to the motor shaft. This may be accomplished by putting a drill chuck on the back shaft. *This technique is applicable only to double-shaft motors with the shaft extending from both ends of the motor.* In extreme cases, you may also need a viscous damper to balance the load. One of the manufacturers of viscous dampers is listed below:

Ferrofluidics Corporation
40 Simon Street
Nashua, NH 03061
(603) 883-9800

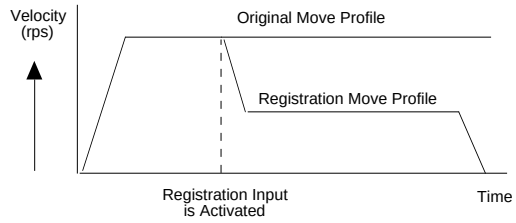
Adjusting the waveform (**MW** command) or changing the velocity (**V** command) and acceleration (**A** command) may also help a resonance problem. Refer to the Tuning Procedures in *Chapter 3, Installation*.

Ringing or Overshoot

The motor's springiness, along with its mass, form an underdamped resonant system that *rings* in response to acceleration transients (such as at the end of a move). Ringing at the end of a move prolongs settling time. *Overshoot* occurs when the motor rotates beyond the actual final position. The actual settling time of a system depends on the motor's stiffness, the mass of the load, and any frictional forces that may be present. By adding a little friction, you can decrease the motor's settling time.

Registration

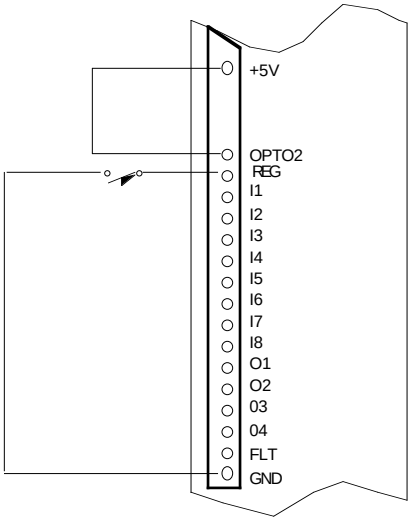
Registration with the SX provides the ability to change the move profile which is being executed to an unrelated move profile defined as a registration move. **This unrelated registration move is executed when an input to the SX transitions from an inactive to active state.** You can only use the input labeled **REG** as a registration input. This input will interrupt the SX microprocessor at the highest priority level in the system. The registration input and the motor's current position will be captured within 50 μs. The registration profile will be executed at the next update period. The registration move uses the actual position captured within 50 μs as its reference point.



Registration Move

The interrupt is *edge-sensitive* to the voltage transition (if you have a bouncy switch for the registration input, you must use the debounce (**TDR**) command to ensure that false registration interrupts do not occur). With the SX, another registration interrupt can interrupt the current registration move. *When the* ***REG input*** *is* **enabled**, *it* ***cannot*** *be toggled at a frequency higher than 1 kHz, or once every ms.*

REG defines the move when the **REG** input is enabled (refer to the figure below). The **INRE** command enables registration, **INRØ** will disable it (see ***SX Software Reference Guide*** for further explanation.)



Partial View of SX

The syntax for defining a registration move using the **REG** input on the front panel is:
REG1,A1Ø,AD1Ø,V1Ø,D25ØØØ

The registration move, **REG1**, will be performed when the **REG** input is activated. The acceleration will be 10 rps^2, the deceleration will be 10 rps^2, the velocity will be 10 rps, and the distance traveled will be 25,000 steps from the captured position. The SX motor must be in motion for a registration move to be performed.

Bouncy Registration Inputs

The registration switch may be bouncy or noisy and may take a few ms to settle. With a bouncy switch, each edge appears like a registration interrupt and the registration move is made from the distance position at which the latest edge was detected. The SX allows you to debounce the inputs (with a software command). You can ignore any bouncing transitions from your switch after the initial registration interrupt has occurred. The time in which the interrupts or false edges are ignored is determined by the number you enter for the **TDR** command. **TDR** is the debounce time in ms that you specify so that the inputs cannot cause another registration interrupt until the switch is settled.

As an example, an application needs a registration move. The motor has a resolution of 25,000 steps per revolution. The registration move must turn the motor 1 revolution at 10 rps. If the input does not occur, the move will be a 500,000-step move at 5 rps. The SX is configured as follows:

Command	Description
> INRE	Enables registration input (REG)
> REG1,A1Ø,AD1Ø,V1Ø,D25ØØØ	Defines the registration move
> D5ØØØØØ	Sets distance to 500,000 steps
> V5	Sets velocity to 5 rps
> G	The preset move is initiated

If **REG** is toggled, the corresponding registration move is performed. The **DIN** command will not activate registration input (the registration input is hardware oriented).

Jogging the Motor

In some applications, you may want to move the motor manually. You can configure the SX to allow you to move the motor manually with the Configure Input (**IN**) command. You can define the jogging velocity with the Jog Velocity High (**JVH**) and Jog Velocity Low (**JVL**) commands. You can define three different inputs for jogging: CW Jog input (**IN#J**), CCW Jog Input (**IN#K**), and Jog Speed Select High/Low (**IN#L**). You must also enable the jogging feature with the **OSE1** command. Once you set up these parameters, you can attach a switch to the jog inputs that you defined and perform jogging (**#** represents digits 1 - 8, which you enter). The following example shows how you can define power-up sequence #100 to set up jogging.

Step ① Define a power-up sequence.

Command	Description
> XE100	Erase sequence #100
> XD100	Define sequence #100
LD3	Disables the limits (*not needed if the limit switches installed*)
JA25	Set jog acceleration to 25 rps^2
JAD25	Set jog deceleration to 25 rps^2
OSE1	Enables Jog function
JVL.5	Sets low-speed jog velocity to 0.5 rps
JVH5	Sets high-speed jog velocity to 5 rps
IN1J	Sets **I1** as a CW jog input
IN2K	Sets **I2** as a CCW jog input
IN3L	Sets **I3** as a speed-select input
XT	Ends sequence definition

Step ② Reset the SX.

Command	Description
> Z	Reset the SX

Step ③ Turn **I1** input on to move the motor CW at 0.5 rps (until you turn off **I1**).

Step ④ Turn **I2** input on to move the motor CCW at 0.1 rps (until you turn off **I2**).

Step ⑤ Turn **I3** input on to switch to high-speed jogging.

Step ⑥ Repeat steps 3 and 4 to perform high-speed jogging.

Changing **I3** during a current jog will change the velocity on the fly.

Backlash Compensation

The SX can compensate for backlash in the gearing of your system. You can specify different compensations depending on the direction the motor is moving. You will use the **BL** command for backlash compensation. Refer to the ***SX Software Reference Guide*** for a detailed description of the backlash command. The command syntax is as follows:

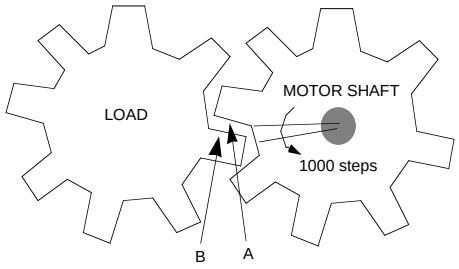
Command	Description
> BLn,m	Variable **n** is the amount of CCW steps that should be compensated. Variable **m** is the number of CW steps that will be compensated.

The **BL** *command is only functional in Motor Step mode*. If a CW move is made, an extra **m** steps will be made to account for backlash. The concept is that the load will not begin to move until the motor moves enough to make contact with the gearing (refer to the figure below). This is also true for CCW moves. The backlash compensation may be different in either direction, so you can program them independently. The load will not move until point A contacts point B. This backlash is associated with changing direction. If the desired distance is 1,000 motor steps, the **BL** command will have the following parameters:

Command	Description
> BL1000,2000	**1000** is the amount of CCW steps that should be compensated. **2000** is the number of CW steps that will be compensated.

For a 25,000-step CCW move, the motor will actually move 26,000 steps to remove the backlash, but the position counter will only change by 25,000 steps. This is done because the load is expected to move whenever the motor moves. The load should move a certain distance when the motor moves a certain distance. If there is backlash in the system, the load will not move the correct distance when the motor is commanded to move in the opposite direction from its previous move. In this example, the motor was moving CW. The gearing was flush and the teeth were touching. When the direction changes, the teeth must move 1,000 steps before they are again in contact with the load gear. Thus, for the load to move 25,000 steps, the motor will have to move 1,000 steps until the teeth are in contact, then move 25,000 steps so that the load will actually move 25,000 steps. This

mode allows you to compensate for the error between the motor and the actual load position. *The compensation only occurs when you change direction.*



Backlash Compensation

Defining a Home Location

The SX's go home function brings your motor to a home reference position. The homing function allows you to back up to a home switch and come to a stop on a specific edge of the switch. You can program the active level of the switch. The homing function also allows you to home to the Z channel of the SX motor's encoder.

Homing to a Switch

There are two homing modes. Normal mode (**OSB0**) is where the motor decelerates from the go home velocity (**GHV**) to a stop, upon the first home input transition, regardless of direction.

Back Up to Home mode (**OSB1**) is where the motor decelerates from the go home velocity to a stop, upon the first home input transition, and either changes direction or continues on at the go home final velocity (**GHF**), looking for another transition of the home switch. This part of the back up to home move is dependant upon the initial homing direction, the final homing direction (**OSG**), and whether the initial deceleration takes you through the other side of the home sensor. **Refer to the included homing diagrams for your needs and set-up parameters.**

A Go Home move is initiated with the **GH** command or with an input defined as a Go Home Input (**INnS**). The **GH** command can also be used to specify the initial homing velocity and direction. These parameters can also be set with the **GHV** command, which does not initiate motion by itself. For example, GH-2 would initiate the go home move in the negative direction at 2 rps. This could also be accomplished with GHV-2 and GH or GHV-2 and a go home input. The **GHA** and **GHAD** commands control the go home acceleration and deceleration. The **OSC** command allows you to adjust the active level of the home input. **Refer to the *SX Software Reference Guide* for further descriptions of all the noted commands.**

Go Home Status Report

The Go Home Status Report (**RG**) can be used to determine if the home move was successful or not.

Homing to a Z Channel

The SX also allows you to home to an encoder's Z channel after locating the home sensor and doing its back up to home move. It is required to see the home input transitions and to be doing a back up to home move in order to home to the Z channel. You do not have to be in encoder mode. Homing to a Z channel is enabled with the **OSD** command and can be configured with the **OSJ** command to search for the Z channel until it is received or through one revolution only. The **RG** command will report back *A, home successful, as long as the home input transitions have been seen. It is independent of whether the Z channel is seen.

Homing and End-of-Travel Limits

Hitting an end-of-travel limit (hardware or software) once during a go home move will cause the motor to reverse direction and seek the desired home transition in the other direction. The home move will then act as though the new direction was the original direction. Hitting the other end of travel limit (hard or soft) in the course of the same go home move will cause the motor to decelerate to a stop. If a home transition has not been found before the two limits are hit, the **RG** command will report back *@, home unsuccessful. However, if a home transition has been encountered before the second limit is hit, the **RG** command will report back ***A**, home successful, so this condition should be avoided for accurate homing. Hitting the second limit after encountering home transitions will not cause a fault (run the fault sequence), since this is considered a successful home.

Homing Diagrams

The following diagrams are examples of the many possible homing set-ups. Your parameters may vary and the results may vary slightly depending on your settings. The following parameters were used for the diagrams shown below.

Set-up Parameters: **GHV1 GHF0.2 GHA1 GHAD1**

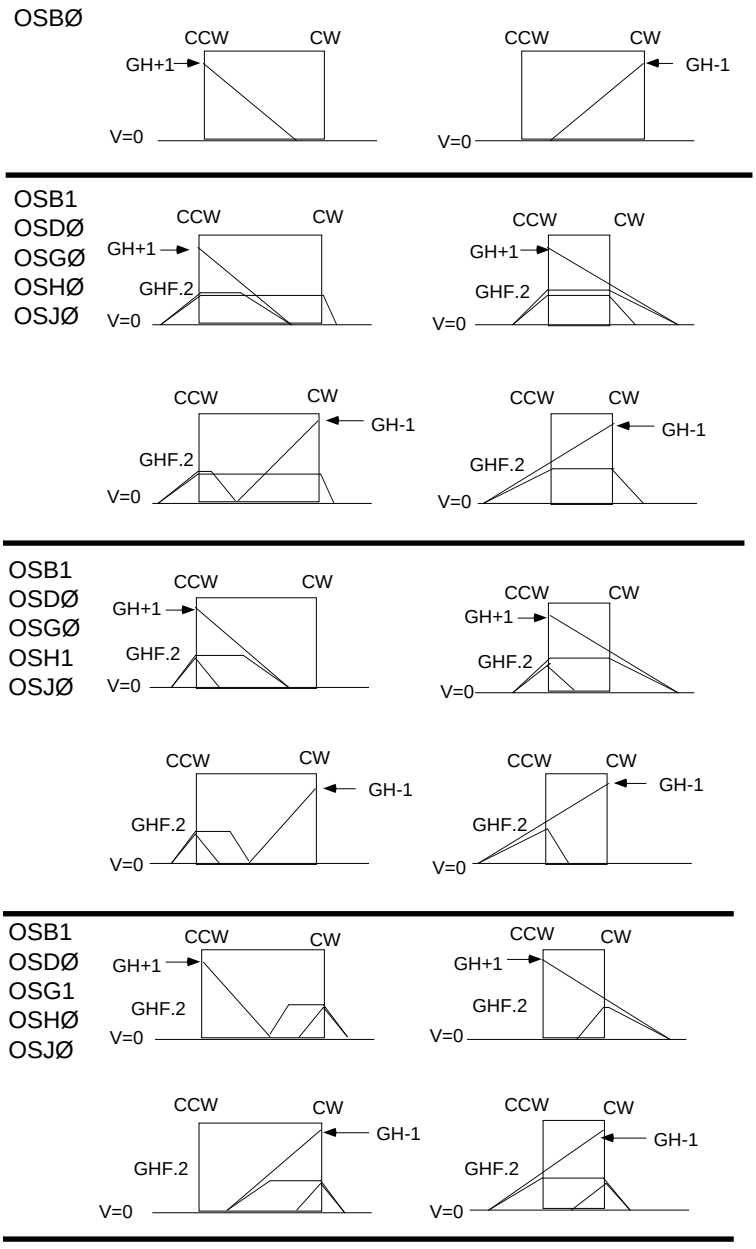
The CW side of the home pulse is the side closest to the CW limit. The CCW side of the home pulse is the side closest to the CCW limit.

The long pulse diagrams are indicative of situations where the motor decelerates while remaining inside the home pulse width due to a rapid homing deceleration or a very wide home pulse. The short pulse diagrams are indicative of situations where the motor decelerates through the home pulse width due to a slow deceleration or a very narrow pulse width.

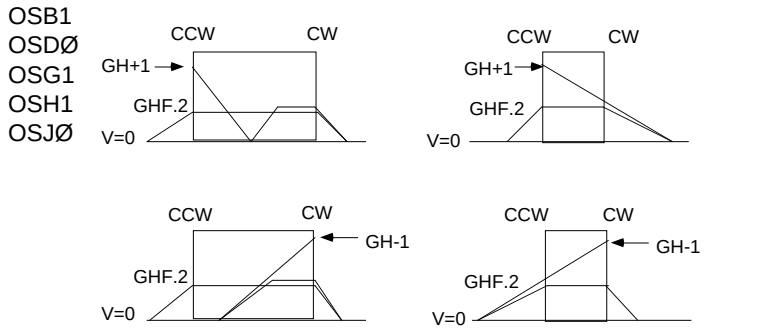
If an end-of-travel limit is hit during the initial homing, refer to the homing diagram for the opposite direction of travel.

✒ Note

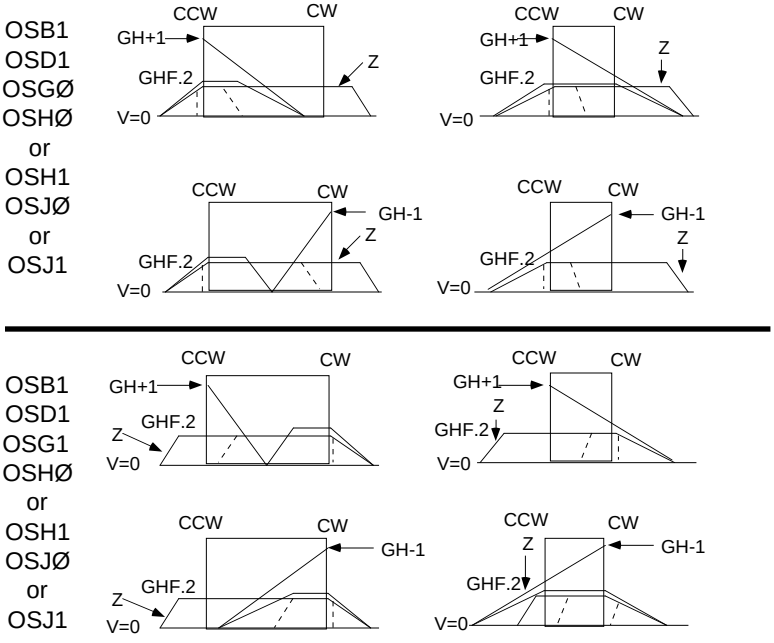
The diagrams are drawn as a general guide. Velocity levels and slopes are drawn to indicate the general move profile the motor will make during the go home move. The vertical axis is velocity and the horizontal axis is position in relation to the home input transitions. Some lines are drawn as closely as possible together to indicate identical velocities yet remain discernible.



Homing Examples



Note: The final stopping point in the diagrams below is dependant upon the location of the encoder's Z channel. As shown, the Z channel can occur anywhere in relation to the home switch, once the final homing move is started.



Homing Examples

Motion Programs and Sequences

You must program the SX to perform motion functions. A motion program typically consists of initialization (or SX set-up), move profiles, and an I/O or RS-232C interface to execute motion instructions. Refer to the ***SX Software Reference Guide*** to determine if a command is buffered or immediate.

Sequence Commands

Sequences are the building blocks of SX motion programs. A sequence can be 1 command or up to 8K bytes of commands. The sequences are stored in battery backed RAM. All commands are followed by a delimiter. You can use either the space character or a carriage return. Either format is acceptable. The following commands define, erase, and run sequences as well as other specialized sequence functions. Refer to the ***SX Software Reference Guide*** for detailed descriptions and syntax of the following commands.

Sequence Status Commands

Command	Description
XBS	Reports the number of bytes available for sequence programming
XC	Sequence checksum report
XDIR	Reports the defined sequences and the bytes of memory they occupy

Sequence Programming Commands

Command	Description
XD	Starts sequence definition
XEALL	Deletes all sequences from battery-backed RAM
XT	Ends sequence definition

Sequence Execution Commands

Command	Description
XQ	Sets/resets interrupted Run mode
XRP	Runs a sequence with a pause
XR	Runs a sequence
SSJ1	Runs a sequence defined by binary weighted sequence inputs

Sequence Branching Commands

Command	Description
XG	Exits current sequence to execute another sequence
GOTO	Exits current sequence to execute another sequence
XR	Jumps to execute another sequence then returns to the originating sequence
GOSUB	Jumps to execute another sequence then returns to the originating sequence

Sequence Debugging Commands

Command	Description
XTR	Sequence Trace mode
XST	Sequence Single Step mode
XS	Sequence Execution status
#	Step sequence command
DIN	Simulate input state command
DOUT	Simulate output state command

Special Sequence Commands

Command	Description
WHEN	Special condition command
XWHEN	Special condition sequence
XFK	Fault sequence

A sequence is a series of commands. These commands are executed in the order in which they are programmed when the sequence is run. Immediate commands cannot be stored in a sequence, just as they cannot be stored in the command buffer. Only buffered commands may be used in a sequence. Refer to the ***SX Software Reference Guide*** to determine if a command is buffered or immediate.

The SX has 8,000 bytes of nonvolatile memory to store 100 sequences. You can use the **XBS** command to determine how many bytes are available in the sequence buffer and the **XDIR** command to determine what sequences have been programmed. The sequence buffers may have variable lengths, so you may have one long sequence or several short ones, as long as the combined total length does not exceed the 8,000 bytes of allocated space.

To begin the definition of a sequence, enter the Define Sequence (**XD**) command immediately followed by sequence identifier number (1 to 100) and a delimiter. The Terminate Sequence (**XT**) command ends the sequence definition. All commands that you enter after the **XD** command and before the **XT** command will be executed when the sequence is run. An example is provided below. Type **1DR** to see the state of the SX.

Command	Description
> 1DR	Displays the present state of the SX.

Perform the following commands:

Command	Description
> MPI	Places the SX in Incremental mode
> MN	Places the SX in Preset mode
> FSIØ	Places the SX in Indexer mode
> LD3	Disables the SX's limits

Command	Description
> XE1	Erases sequence #1
> XD1	Begins definition of sequence #1
A25	Sets acceleration to 25 rps^2
AD25	Sets deceleration to 25 rps^2
V1Ø	Sets velocity to 10 rps
D5ØØØ	Sets distance to 5,000 steps
G	Executes the move (Go)
H	Reverses direction
G	Executes the move (Go)
XT	Ends definition of sequence
> XR1	Runs sequence #1

You can run a sequence by entering the **XR** command immediately followed by a sequence identifier number (1 - 100) and a delimiter. **Once you define a sequence, it cannot be redefined until you delete it.** You can delete a sequence by entering the **XE** command immediately followed by a sequence identifier (1 - 100) and a delimiter. You may then redefine that sequence. *You can use* **XEALL** *to delete all defined sequences from battery-backed RAM*. **Use these commands with extreme caution. Erased sequences cannot be retrieved**.

Sequence #100 is a power-up sequence (if you have defined it). It is always run when you power up the system or when you reset the Indexer with the Reset (**Z**) command. For convenience, you may find it advantageous to place all of your set-up commands in sequence #100. Sequences that you define are automatically saved into the SX's nonvolatile memory. The only way to erase these sequences individually is by using the Erase Sequence (**XE**) command.

Creating and Executing Sequences

Sequences can be created via RS-232C serial communications. Before you create sequences, you must understand the types of motion and the required user interfaces. To determine the proper user interface, you should be familiar with the methods of selecting sequences within your application.

Selecting Sequences

After you define the sequences from the RS-232C interface, you can execute the sequences by using one of the following modes of operation:

- ❏ Stand Alone: Select sequences using discrete inputs or thumbwheels See, *Stand Alone Operation* in this chapter for an example of this feature.
- ❏ Computer Interface: Use the Execute Sequence (**XR**) command to run sequences. See, *Stand Alone Operation* in this chapter for an example.
- ❏ PLC (Programmable Logic Controller): Use the sequence select inputs to run sequences. See *Stand Alone Operation* in this chapter for an example.

Subroutines

When you use the **GOTO** Sequence (**XG**) and the Run a Sequence (**XR**) commands, you can execute new sequences from within the current sequence. These commands can replace the **GOTO** and **GOSUB** commands respectively. If you use **XG** or **GOTO**, the program will move to the specified sequence. After executing the specified sequence, the system will not return to the original sequence. It will remain in the current sequence, unless it receives another execution command (**XG/GOTO** or **XR/GOSUB**). However, if you use **XR** or **GOSUB**, the program will return control to the original sequence that contained **XR** or **GOSUB**, when **XT** is reached in the subroutine. This prompts the program to return to the sequence that initiated the move to another sequence.

You can nest as many as 16 different levels of sequences within one program. For example, when you exit sequence #1 to execute sequence #2 with **XR2**, you could **GOSUB** to sequence #3 from sequence #2. This procedure of nesting **XR** commands can be repeated 16 times. The **XG** command has no nesting limit since the program will not return control to the original sequence.

Command	Description
> XE2	Erases sequence #2
> XD2	Defines sequence #2
A1ØØ	Sets acceleration to 100 rps^2
AD1ØØ	Sets deceleration to 100 rps^2
V5	Sets velocity to 5 rps
D25ØØØ	Sets distance to 25000 steps
G	Executes the move (Go)
XT	Ends sequence #2 definition
> XE3	Erases sequence #3
> XD3	Defines sequence #3
A1Ø	Sets acceleration to 10 rps^2
V5	Sets velocity to 5 rps
D-25ØØØ	Sets distance to 25000 steps (CCW direction)
G	Executes the move (Go)
XT	Ends sequence #3 definition
> XE1	Erases sequence #1
> XD1	Defines sequence #1
XR2	Executes sequence #2
GOSUB3	Executes sequence #3 (same as an **XR** command)
XT	Ends sequence #1 definition
> 1XTR1	Enables the Trace mode
> XR1	Executes sequence #1

In the previous example, when you execute sequence #1, the program moves to sequence #2. After executing sequence #2, the program returns to sequence #1. The program then moves to execute sequence #3. The Trace mode (**XTR1**) was enabled to help you see how the sequence was executed.

Asynchronous Events—FAULT and WHEN

The SX has special sequences that can be run when a certain condition occurs.

- The power-up sequence has already been explained (sequence #100). This sequence is always run on power up and is often used to store the set-up commands for the application.
- **FAULT** sequence
- **WHEN** sequence

FAULT Sequence

The fault sequence is a sequence that is automatically executed when a fault condition or a Kill (**K**) occurs. Any condition that causes the SX to fault invokes the fault sequence (if one is defined). Refer to the list below for conditions that will prompt execution of the fault or kill sequence. A sequence is designated as a fault or kill sequence with the **XFK** command. You can use the fault sequence to place the SX in a safe state and turn off outputs that may be harmful to the rest of the system. You can use an **IF** command to determine what condition caused the fault so that the appropriate action can be performed. The **IF** statement is explained in the next section. The following steps illustrate the use of a fault sequence. The **IF** statement section explains more about the **ERnnnnn** flag that indicates what fault condition occurred.

Example

Step ①

Define an input as a user fault input. You can use this input to indicate that a fault has occurred somewhere external to the system. This input will cause a fault condition.

Command	Description
> IN1U	Defines input #1 as a user fault input

Step ②

Designate sequence #10 as the fault sequence.

Command	Description
> XFK1Ø	Designates sequence #10 as the fault sequence
> XE1Ø	Erases sequence #10
> XD1Ø	Defines sequence #10
1"External_Fault	Quote command
XT	End definition of sequence #10

This defines the fault sequence. The quote command sends a message over the RS-232C link to the terminal to tell the operator that a fault has occurred. You can use the Quote command to write statements to the terminal. Remember, the space character is a delimiter so the underscore character should be used to separate words in the quote command. Sequence #10 will now be executed whenever a fault occurs. In the following example, an alternating loop will be performed.

Step ③

The normal *state* of this example application will be an alternating loop.

Command	Description
> A5Ø	Acceleration is 50 rps^2
> AD4Ø	Deceleration is 40 rps^2
> D5ØØØØ	Distance is 50000 steps
> V7	Velocity is 7 rps
> L	Loop command
G	Initiates motion
H	Change direction
N	End the loop

The motor will alternate back and forth continuously.

Step ④

You will now cause a system user fault error. Input states can be simulated using the **DIN** command. Type the following to simulate the activation of the user fault input.

Command	Description
> DINEEEE1	Input #1 is activated

The user fault should have occurred and sequence #10 should have automatically been executed when the fault occurred. To clear the fault from the display, enter the following command.

Command	Description
> DINEEEEØ	Input #1 is de-activated

An **IF** statement could have been used in the fault sequence to determine what fault condition occurred then branch to a sequence that handled that fault condition appropriately. For example, the fault sequence will run for several faults. These faults are limits reached, general product faults, and user faults. Each bit in the error flag (**ERnnnn**) will correspond to one of these fault conditions. Use the **IF** statement to determine which one occurred. Retype Sequence #10 (the fault sequence) as follows:

Command	Description
> 1XE1Ø	Erases sequence #10
> 1XD1Ø	Defines sequence #10
IF(ER1)	If CCW limit is hit,
1"CCW_LIMIT_WAS_HIT	
NIF	End the **IF (ER1)** statement
IF(ERX1)	If CW limit is hit,
1"CW_LIMIT_WAS_HIT	
NIF	End the **IF (ERX1)** statement
IF(ERXXXXXX1)	If user fault occurs
1"USER_fault	
NIF	Ends the **IF (ERXXXXXX1)** statement
XT	End fault sequence statement

Depending on which error caused the program to branch to the fault sequence, the appropriate message will be displayed. Before continuing with the user guide, disable the fault sequence with the **XFKØ** command.

List of Fault or Kill Sequence Causes

- ❏ Over Temperature Fault (motor de-energized)
- ❏ Motor Fault (motor de-energized)
- ❏ Under Voltage Fault (motor de-energized)
- ❏ Commanded Motor Shutdown (motor de-energized)
- ❏ End-of-Travel Limit Hit (hard or soft) (motor not de-energized)
- ❏ Excessive Position Error (motor not de-energized)
- ❏ Absolute Encoder Rollover (motor not de-energized)
- ❏ User Fault Input Active (motor de-energized)
- ❏ Commanded Kill (**K** command) (motor not de-energized)
- ❏ Kill Input Transition (motor not de-energized)

WHEN Sequence

The **WHEN** sequence is a sequence that can be designated to run when a specific condition evaluates true. This could be a variable having a certain value, the inputs being in a specific state, or the user flag having a set state. Whatever sequence or program is currently being run will be interrupted when the condition is true and the sequence designated by the **XWHEN** command will be executed. There are certain commands which will not be interrupted by the **WHEN** condition. These are **G**, **GH**, **TR**, **U**, **PS**, or a jogging move. The **XWHEN** sequence can be executed during a **G** move or a jogging move if in Position Profile mode (**MPP**). However, the move itself will not be stopped automatically. In the following example, when variable 3 is greater than 50, or Input #1 is on, the SX will execute the **XWHEN** sequence.

Command	Description
> WHEN(VAR3>5Ø_OR_INXXXX1)	Defines the **WHEN** statement that must be true in order for the **XWHEN** sequence to be run.

Step ①

The **WHEN** sequence is now defined. Sequence #8 is designated as the **WHEN** sequence.

Command	Description
> XWHEN8	Sequence #8 is designated as a **WHEN** sequence
> XE8	Erases sequence #8
> XD8	Defines sequence #8
A5Ø	Acceleration is 50 rps^2
AD4Ø	Deceleration is 40 rps^2
D3ØØØØ	Distance is 30000 steps
V5	Velocity is 5 rps
G	Initiates motion
XT	Ends the sequence definition

Step ②

The normal program to be executed is defined and executed here.

Command	Description
> VAR3=Ø	Variable 3 is initialized to Ø
> A5Ø	Acceleration is 50 rps^2
> AD4Ø	Deceleration is 40 rps^2
> D5ØØØØ	Distance is 50000 steps
> V7	Velocity is 7 rps
> L	Loop command
G	Initiates motion
VAR3=VAR3+1	Variable 3 is increased
N	End the loop

Step ③

You can activate the **WHEN** sequence in two ways.

- ❑ The **DIN** command can activate the input that will satisfy the **WHEN** condition.
- ❑ The current program can be run 50 times, then the **WHEN** sequence will be run.

Step ④

Using an input to cause program execution to jump to the **XWHEN** sequence can change or interrupt the program. Within the **XWHEN** sequence, you can use an **IF** statement to check the state of one or more inputs. Based on the state of the inputs, different sequences can be executed. **Disable the XWHEN sequence before continuing with this procedure by entering XWHENØ.**

Power-Up Sequence Execution

You can program the SX to execute a sequence of commands on power up (sequences can be used as subroutines). Sequence #100 always runs on power up. To run another sequence on power up, put an **XR**<num> (or **XG**<num>) at the end of sequence #100. If sequence #100 is empty, nothing happens on power up. Refer to the ***SX Software Reference Guide*** for detailed descriptions and syntax of the following commands.

Command	Description
> XE1ØØ	Erases sequence #100
> XD1ØØ	Begins definition of sequence #100
LD3	Disables limits
A2Ø	Sets acceleration to 20 rps^2
AD2Ø	Sets deceleration to 20 rps^2
V5	Sets velocity to 5 rps
D125ØØ	Sets distance to 12,500 steps
MN	Sets SX in Preset mode
MPI	Sets SX in Incremental mode
FSIØ	Sets unit to the Indexer mode (non Follower)
XG1	Go to sequence #1
XT	Ends sequence definition
> XE1	Erases sequence #1
> XD1	Defines sequence #1
G	Executes a go command
XT	Ends sequence definition
> Z	Resets the Indexer and runs sequence #100

A power-up sequence is typically used to store set-up or initialization parameters that your application requires. Having motion in your power up sequence is not recommended. Examples of these set up commands are listed below.

Command	Description
SSJ1	Continuous Sequence Scan Mode
SN	Scan time
JA	Jog acceleration
JVL	Jog velocity low
JVH	Jog velocity high

SS, **FS**, **OS**, **IN**, and **OUT** commands are examples of set-up commands typically put into sequence #100. You can put any buffered commands into sequence #100 (if you want to execute them during power up).

Sequence Debugging Tools

After creating your sequences, you may need to debug them to ensure that they are performing properly. The SX provides several debugging tools.

- ❏ In Trace mode, you can trace a sequence as it is executing.
- ❏ You can set the desired state of the SX 's I/O via software commands.
- ❏ Error messages can be enabled explaining why the SX has stopped execution due to a programming error.

Trace Mode

You can use the Trace mode to debug a sequence or a program of sequences. The Trace mode allows you to track, command-by-command, the entire sequence. It displays to your terminal, over the RS-232C serial link, all of the commands as they are executed. The following example demonstrates the Trace mode.

Step ① Create the following sequence:

Command	Description
> XE1	Erases sequence #1
> XD1	Begins definition of sequence #1
A1Ø	Acceleration is 10 rps^2
AD1Ø	Deceleration is 10 rps^2
V5	Velocity is 5 rps
L5	Loop 5 times
GOSUB3	Jump to sequence #3
N	Ends the loop
XT	Ends the definition of sequence #1

Step ② Define sequence #3.

Command	Description
> XE3	Erases sequence #3
> XD3	Begins definition of sequence #3
D5ØØØØ	Sets the distance to 50000 steps
G	Initiates motion
XT	Ends the definition of sequence #3

Step ③ Enter the following command to enable the Trace mode.

Command	Description
> 1XTR1	Enables the Trace mode

Step ④ You will now execute the sequence. The commands will be displayed on the terminal as each command in the sequence is run. Enter the following command.

Command	Response
> XR1	Run sequence #1—response:
*SEQUENCE_ØØ1	COMMAND_A1Ø
*SEQUENCE_ØØ1	COMMAND_AD1Ø
*SEQUENCE_ØØ1	COMMAND_V5
*SEQUENCE_ØØ1	COMMAND_L5
*SEQUENCE_ØØ1	COMMAND_GOSUB3____LOOP_COUNT_1
*SEQUENCE_ØØ3	COMMAND_D5ØØØØ____LOOP_COUNT_1
*SEQUENCE_ØØ3	COMMAND_G____LOOP_COUNT_1
*SEQUENCE_ØØ3	COMMAND_XT____LOOP_COUNT_1
*SEQUENCE_ØØ1	COMMAND_N____LOOP_COUNT_1
*SEQUENCE_ØØ1	COMMAND_GOSUB3____LOOP_COUNT_2
*SEQUENCE_ØØ3	COMMAND_D5ØØØØ____LOOP_COUNT_2
*SEQUENCE_ØØ3	COMMAND_G____LOOP_COUNT_2
*SEQUENCE_ØØ3	COMMAND_XT____LOOP_COUNT_2
*SEQUENCE_ØØ1	COMMAND_N____LOOP_COUNT_2
*SEQUENCE_ØØ1	COMMAND_GOSUB3____LOOP_COUNT_3
*SEQUENCE_ØØ3	COMMAND_D5ØØØØ____LOOP_COUNT_3
.	
.	
.	
*SEQUENCE_ØØ3	COMMAND_G____LOOP_COUNT_5
*SEQUENCE_ØØ3	COMMAND_XT____LOOP_COUNT_5
*SEQUENCE_ØØ1	COMMAND_N____LOOP_COUNT_5
*SEQUENCE_ØØ1	COMMAND_XT

The format for the Trace mode display is: **Sequence Number_Command_Loop Count**

Step ⑤ To exit the Trace mode, enter the following command:

Command	Description
> 1XTRØ	Exits Trace mode

Single-Step Mode

You can debug your program with another level of debugging with the Single-Step mode. Single-Step mode allows you to execute one command at a time when you want the command to be executed. Use the **XST** command to enable Single-Step mode. Once you are in the mode, you can execute a sequence one command at a time. To execute a command, you must use the **#** sign. By entering a **#** followed by a delimiter, you will execute the next command in the sequence. If you follow the **#** sign with a number (*n*) and a delimiter, you will execute the next *n* commands. To illustrate Single-Step mode, use the following steps:

Step ① Enter Single-Step mode and Trace mode.

```
> XST1
> 1XTR1
```

Step ② Begin execution of sequence #1.

```
> XR1
```

Step ③ You will not execute any commands until you use the # command. Type the following.

Command	Description
#	Executes one command

The response will be:

```
*SEQUENCE_ØØ1____COMMAND_A1Ø
```

Step ④ To execute more than one command at a time follow, the # sign with the number of commands you want executed.

Command	Description
#3	Executes 3 commands, then pauses sequence execution

To complete the sequence, use the **#** sign until all the commands are completed. To exit Sequence-Step mode, type:

```
> XSTØ
```

Simulating I/O Activation

If your application has inputs and outputs that integrate the SX with other components in your system, you can simulate the activation of these inputs and outputs so that you can run your sequences without activating the rest of your system. Thus, you can debug your program independently of the rest of your system. This is the same way in which a PLC program can be debugged by simulation of input and output states to run various portions of the program. The SX uses two commands that allow you to simulate the input and output states desired. The **DIN** command controls the inputs and the **DOUT** command controls the programmable outputs.

You will generally use the **DIN** command to cause a specific input pattern to occur so that a sequence can be run. Use the **DOUT** command to simulate the output patterns that are needed to prevent an external portion of your system from operating. You can set the outputs in a state that will be the inactive state of your external system. When you execute your program, a part in the program that will activate the outputs will not actually turn the outputs on to their active state because the **DOUT** command overrides this output and holds the external portion of the machine in an inactive state. When the program is running smoothly without problems you can activate the outputs and the SX will affect the external system.

Outputs

The following steps describe the use and function of the **DOUT** command.

Step ①

Display the state of the outputs with the **OUT** command and the **O** command.

Command	Description
> 1OUT	Displays the state of the outputs
The response will be:	
*1_A_PROGRAMMABLE_OUTPUT_____	(STATUS_OFF)
*2_A_PROGRAMMABLE_OUTPUT_____	(STATUS_OFF)
*3_A_PROGRAMMABLE_OUTPUT_____	(STATUS_OFF)
*4_A_PROGRAMMABLE_OUTPUT_____	(STATUS_OFF)
*5_DEDICATED_FAULT_OUTPUT_____	(STATUS_OFF)

Command	Response
> 1O	*ØØØØØ
> DOUT11EE	

Step ②

Change the output state using the **O** command.

Command	Description
> O111Ø	

Step ③

Display the state of the outputs with the **OUT** command and the **O** command.

Command	Description
> 1OUT	Displays the state of the outputs
The response will be:	
*1_A_PROGRAMMABLE_OUTPUT	(DISABLED_ON)
*2_A_PROGRAMMABLE_OUTPUT	(DISABLED_ON)
*3_A_PROGRAMMABLE_OUTPUT	(STATUS_ON)
*4_A_PROGRAMMABLE_OUTPUT	(STATUS_OFF)
*5_DEDICATED_FAULT_OUTPUT	(STATUS_OFF)

Command	Response
> 1O	*111Ø

Step ④

You can now disable the outputs into the inactive state using the **DOUT** command. An **E** does not affect the output.

> DOUT00EE

Step ⑤

Display the state of the outputs with the **OUT** command and the **O** command.

Command	Description
> 1OUT	Displays the state of the outputs
The response will be:	
*1_A_PROGRAMMABLE_OUTPUT	(DISABLED_OFF)
*2_A_PROGRAMMABLE_OUTPUT	(DISABLED_OFF)
*3_A_PROGRAMMABLE_OUTPUT	(STATUS_ON)
*4_A_PROGRAMMABLE_OUTPUT	(STATUS_OFF)
*5_DEDICATED_FAULT_OUTPUT	(STATUS_OFF)

Command	Response
> 1O	*ØØ1Ø

Inputs

The following steps describe the use and function of the **DIN** command. You can use it to activate an input state. The inputs will not actually be in this state, but the SX treats them as if they are in the given state and will use this state to execute its program.

Step ①

This sequence will wait for a trigger state to occur and will then begin moving in Continuous mode. An input that is configured as a stop (**S**) input will stop motion.

Command	Description
> 1IN1A	Input #1 is a trigger input
> 1IN2A	Input #2 is a trigger input
> 1IN3D	Input #3 is a stop input
> 1INLØ	The active input level is low
> 1XE1	Erase sequence #1
> 1XD1	Define sequence #1
TR11	Waits for the input trigger state to be 11
A1ØØ	Acceleration is set to 100 rps^2
AD1ØØ	Deceleration is 100 rps^2
V5	Velocity is 5 rps
MC	The Continuous mode is activated
TR1Ø	Waits for a trigger input state of 10
G	Begins motion
XT	Ends sequence definition

Step ②

Turn on Trace mode so that you can view the sequence as it is executed.

Command	Description
> 1XTR1	Turns on the Trace mode

Step ③

Execute the sequence.

Command	Description
> XR1	Runs sequence #1

Step ④

The sequence will execute until the **TR11** command is encountered and will then pause waiting for the trigger condition to be satisfied. Simulate the input state using the **DIN** command. Inputs with an **E** value are not affected.

```
1DINEEEE11EEEEE
```

Step ⑤

The sequence will now execute until the **TR1Ø** trigger is encountered and will pause waiting for the new input pattern. Use the **DIN** command to simulate the desired input state.

```
1DINEEE1ØEEEEE
```

Step ⑥

The motor will now move continuously until a the stop input is activated. Activate the stop input with the **DIN** command.

```
> 1DINEEEEEE1EEEE
```

The motor will stop. This illustrates the use of the **DIN** command to simulate the activation of input commands. To deactivate the I/O simulation commands, type the following commands:

```
> 1DINEEEEEEEEEE
> 1DOUTEEEE
```

Error Messages

The SX has an Error Message mode that can display error messages when an invalid command is attempted. This error message can be useful in debugging a sequence to determine what was wrong with the command that caused the sequence to not run properly. **SSN1** enables Error Message mode.

Command	Description
> 1SSN1	Enters Error Message mode
> 1D1ØØØØØØØØØØØ	*Invalid data field (distance specified is too large)*

The SX will respond with an error message: ***INVALID_DATA_FIELD**

Command	Description
> 1SSN1	Enters Error Message mode
> 1DK1ØØØØ	*Invalid command (DK is not a valid X Series command)*

The SX will respond with an error message: ***INVALID_COMMAND**

SSNØ disables Error Message mode.

Command	Description
> 1SSNØ	Exits the error message mode

The Model SX's X-language includes several commands that are common in most high-level programming languages (Pascal, Fortran or BASIC). In addition to these commands, 50 variables **VAR1-VAR5Ø** are provided for performing mathematical functions and boolean comparisons. Along with these variables are certain system variables that you can access—**POS** (Commanded Position or Setpoint), **FEP** (Following Encoder Position) or **ABS** (Actual Encoder Position). This section will introduce and explain how to use these structures with the SX.

Variables

The SX has up to 50 variables that can be used to perform multiplication, division, addition, and subtraction. You can assign these variables to various motion parameters. These parameters and the syntax of assigning a variable to them are listed here.

Command	Description
D(VAR1)	Loads the distance with the value of variable 1
V(VAR4)	Loads the velocity with the value of variable 4
A(VAR5)	Loads the acceleration with the value of variable 5
AD(VAR7)	Loads the deceleration with the value of variable 7
FP(VAR9)	Loads the following port with the value of variable 9
DP(VAR1)	Loads the distance point with the value of variable 1
L(VAR3Ø)	Loads the loop count with the value of variable 30
XR(VAR21)	Executes the sequence number held in variable 21
T(VAR3)	Loads the time delay with the value of variable 3
FOL(VAR29)	Loads the following ratio with the value of variable 29

Variable assignments can be made in sequences or in Immediate Terminal mode. If a variable that has a fractional portion is assigned to a parameter that requires a whole number (such as distance), only the whole number portion of the variable gets assigned to the parameter. As an example of the use of the variables and the math functions they can perform, complete the steps below.

Assigning Variables to Constants

You can set up parameters as variables in a sequence (**D(VAR)**). You can define these variables by assigning a constant value. When the sequence is executed, this value is assigned to the corresponding parameter in the sequence.

Step ①

The sequence below executes a move and travels the distance provided in variable 1 and the velocity provided in variable 2. Enter the following commands.

Command	Description
> LD3	Disables limits (*Not needed if limits are installed*)
> MPI	Places the SX in Incremental mode
> MN	Enters the Normal mode
> FSIØ	Sets to Indexer mode (not needed if you do not have Following option—SXF)
> XE1	Erases sequence #1
> XD1	Defines sequence #1
A1Ø	Sets Acceleration to 10 rps^2
AD1Ø	Sets deceleration to 10 rps^2
V(VAR2)	Assigns variable 2 to the velocity term
D(VAR1)	Assigns variable 1 to the distance term
G	Initiates motion
XT	Ends the sequence definition

Step ②

You will now assign numbers to the variables **VAR1** and **VAR2**.

Command	Description
> VAR1=25ØØØ	The distance parameter is assigned 25,000
> VAR2=5	The velocity parameter is assigned 5

Step ③ Execute sequence #1 with (**XR1**). The motor will move 25000 steps at 5 rps. Verify that the distance (**D**) and the velocity(**V**) have been assigned these values.

Command	Response
> 1V	*V05.00000
> 1D	*D+0000025000

Step ④ Now change the values of the variables as follows.

Command	Description
> VAR1=75000	The distance parameter is 75,000
> VAR2=10	The velocity parameter is assigned 10

Step ⑤ Repeat steps 3 and 4.

Variables Via RS-232C

When using variables in sequences, the **RSIN** command is a useful tool for interactively prompting the user to enter a variable number. An example of **RSIN** is given in the following steps.

Command	Description
> 1XE1	Erases current sequence #1
> 1XD1	Defines sequence #1
1"ENTER_THE_NUMBER_OF_PARTS	Prompts you for the # of parts to make
1CR	Inserts a carriage return
1LF	Inserts a line feed
VAR5=RSIN	Sequence execution stops until you enter a #
1CR	Inserts a carriage return
1LF	Inserts a line feed
L(VAR5)	The loop count is loaded with the # of parts to make
D25000	Distance is 25000 steps
G	Initiates motion
N	Ends the loop
1"FINISHED_WITH_PARTS_RUN	Indicates that the run is finished
1CR	Carriage return
1LF	Line feed
XT	Ends definition of sequence #1

Entering The **RSIN** command prompts you to enter a variable via RS-232C that will be used in a sequence. When a variable is assigned to **RSIN**, the execution of the sequence is stopped until you enter the variable . To enter the variable, you must precede the number with an exclamation point (**!**).

Step ① Turn on the Trace mode and run the sequence.

```
> 1XTR1
> XR1
```

Step ② The program will stop at the **RSIN** command and wait for you to be enter a variable number. Enter the variable number.

```
> !10
```

The sequence executes a 25,000-step move 10 times making 10 parts.

Math Operations

In addition to assigning constants to variables, three system parameters can be assigned to a variable.

- **POS**—Commanded Position
- **FEP**—Following Encoder Position
- **ABS**—Actual Encoder Position

These are *read-only* variables. They allow you to read commanded positions, following encoder positions, or actual encoder positions at any time by setting a general-purpose variable equal to the read only variable. They can be also be used for conditional comparisons (i.e., **IF(POS>25000)**).

Command	Response
> VAR1=POS	Assigns the current value of the command position to variable #1

Performing Math Operations with Variables

The SX has the ability to perform simple math functions with its variables (add, subtract, multiply, and divide). The following sequence of steps illustrates the math capabilities of the SX. Note; Only one math function can be performed per command line (i.e., **VAR1=4*5+6** is illegal).

Step ①

Addition:

Command	Response
> VAR1=5	
> VAR23=1000.565	
> VAR11=VAR1+VAR23	
> VAR23=VAR11+VAR1	
> VAR1=VAR1+5	
> 1VAR1	*+0000000000000010.00000
> 1VAR11	*+0000000000001005.56500
> 1VAR23	*+001010.56500
> VAR3=POS+25000	
> 1VAR3	(Commanded Position plus 25000)

Step ②

Subtraction:

Command	Response
> VAR3=10	
> VAR20=15.5	
> VAR3=VAR3-VAR20	
> VAR19=ABS-25000	
> 1VAR3	*-0000000000000005.50000
> 1VAR20	*+0000000000000015.50000
> 1VAR19	(Actual Encoder Position minus 25000)

Step ③

Multiplication:

Command	Response
> VAR3=10	
> VAR20=15.5	
> VAR3=VAR3*VAR20	
> VAR19=ABS*POS	
> 1VAR3	*+0000000000000155.00000
> 1VAR20	*+0000000000000015.50000
> 1VAR19	(Actual Encoder Position multiplied by commanded position)

Step ④

Division:

Command	Response
> VAR3=10	
> VAR20=15.5	
> VAR3=VAR3/VAR20	
> VAR30=75	
> VAR19=VAR30/VAR3	
> 1VAR3	*+0000000000000000.64516
> 1VAR20	*+0000000000000015.50000
> 1VAR30	*+0000000000000075.00000
> 1VAR19	*+0000000000000116.25023

Data

Inputs can be defined as data inputs to allow for external entry of motion data, loop counts, sequence select, time delays, and variable values. The following commands will read and enter data from the inputs:

- ❑ **VARD** Variable Read
- ❑ **DRD** Distance
- ❑ **VRD** Velocity
- ❑ **LRD** Loop Count
- ❑ **TRD** Time Delay
- ❑ **XRD** Sequence Number
- ❑ **FRD** Following Ratio

The weighting for data inputs is *binary coded decimal* or BCD. This weighting allows you to enter data via thumbwheels. To use the data inputs to enter data, the outputs must also be used. Outputs 1 - 3 must be configured as data strobe outputs. They are used to select or strobe the appropriate digit while reading data. Up to 16 digits of data and one sign bit may be entered. The recommended and most common method of controlling the input lines to read data is through thumbwheels. A PLC may also be used to enter data. An explanation of interfacing to thumbwheels or a PLC is introduced later in this chapter.

Delays

You can use the Time (**T**) command to halt the operation of the Indexer function for a preset time. If you are in the Continuous mode and Position Profile Mode, you may use the Time (**T**) command to run the motor at continuous velocity for a set time, then change to a different velocity. In Preset mode, the motor finishes the move before the Indexer executes the time delay.

Command	Description
> PS	Waits for the SX to receive a **C** command before executing the next command
D25ØØØ	Sets distance to 25000 steps
G	Moves motor 25,000 steps
T5	Waits 5 seconds after the move ends
H	Changes motor direction
G	Moves motor 25,000 steps in the opposite direction
C	Continues execution

Complex Branching and Looping

The SX supports the high-level language structures for branching and looping. Each conditional branch or loop evaluates a condition statement. Depending on whether this condition statement evaluates true or not determines where the SX will branch to. The unconditional branching and looping statements have been introduced already. These are the **GOTO** (Branch), **GOSUB** (Branch & Return) and **L** (Loop) command. The **L** command is explained further here.

Unconditional Looping

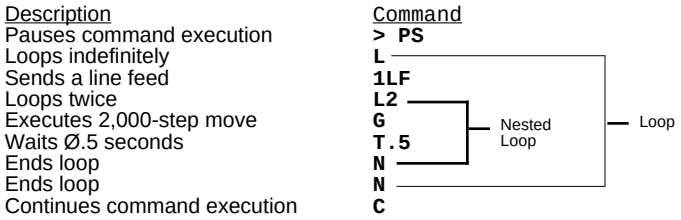
The Loop (**L**) command is an unconditional command. You may use **L** to repeat a series of commands. You can nest loop commands up to 16 levels deep.

Command	Description
> PS	Pauses command execution until the SX receives a **C** command
MPI	Sets unit to Incremental mode
A5Ø	Sets acceleration to 50 rps^2
V5	Sets velocity to 5 rps
L5	Loops 5 times
D2ØØØ	Sets distance to 2,000 steps
G	Executes the move (Go)
T2	Delays 2 seconds after the move
N	Ends loop
C	Continue—Initiates command execution to resume

The example below shows how to nest a loop inside a loop. The motor makes two moves and returns a line feed. The unit repeats these procedures until you instruct it to stop.

☛ **Helpful Hint:**

This command execution continues until you issue the Stop (**S**) or Kill (**K**) commands.



Unconditional Branching

The unconditional branching commands **GOTO** and **GOSUB** were explained earlier in the sequence section.

Conditionals

The branching commands evaluate condition statements to make branching decisions. If the condition is true, one set of commands is processed. If the condition is false, another set of commands may be executed. The commands that evaluate conditions are listed below.

IF (condition true)—*execute these commands*
ELSE—*execute these commands*
NIF

WHILE (condition true)—*execute these commands*
NWHILE

REPEAT—*execute these commands*
UNTIL (condition true)

Condition statements can be very complex.

You can use the following types of conditional statements with the SX.

- ❑ Error Flags
- ❑ User Flags
- ❑ Input State
- ❑ Variable Comparisons
- ❑ Boolean Comparisons

Error Flags

The error flag (**ERXXXXXXXX**) is useful if you want to trap different error conditions and create different sequences to respond to them. The ***SX Software Reference Guide*** explains the errors that can be trapped in the evaluation command description. An example of the statement's use is provided below.

```
> IF(ER110XXXX)
> GOSUB2
> NIF
```

User Flags

You can set the user flag (**FLØØØ111XØ**) and modify it within sequences to mark where the program has gone or to indicate any special state so that a conditional statement can be made. An example of the statement's use is provided below. The **SFL** command is used to set the user flag.

```
> WHILE(FLØØ11XXXX)
> D1ØØ
> G
> NWHILE
```

Input State

An example of this statement's use (**IN1111ØØØ1Ø**) is provided below.

```
> REPEAT
> GOSUB4
> UNTIL(INØØ1XX11XXXXX)
```

Variable Comparisons

Typical variable comparisons are shown below.

```
> VARn>VARm
> VARn<VARm
> VARn=VARm
> IF(VAR1>VAR2)
> WHILE(VAR3=1Ø)
> GOSUB2
> NWHILE
> NIF
> IF(VAR1>POS)
> GOSUB3
> NIF
> REPEAT
> IF(INXXXX1)
> GOTO3
> NIF
> UNTIL(POS>1ØØØØØ)
```

Boolean Comparisons

AND
OR

An example of the statement's use is provided below.

```
>WHILE(VAR3>1Ø_AND_IN11ØØ1_OR_VAR4=1)
>GOSUB8
>NWHILE
```

Condition statements are explained in the ***SX Software Reference Guide***.

Conditional Looping

The SX supports two conditional looping structures—**REPEAT/UNTIL** & **WHILE/NWHILE**.

REPEAT/ UNTIL

With the **REPEAT** command, all commands are repeated between **REPEAT** and **UNTIL**, until the condition is true. Enter the following sequence.

Step ①

Command	Description
> VAR5=Ø	Initializes variable 5 to 0
> 1XE1Ø	Erases sequence #10
> 1XD1Ø	Defines sequence #10
REPEAT	Begins the **REPEAT** loop
A5Ø	Acceleration is 50 rps^2
AD5Ø	Deceleration is 50 rps^2
V5	Sets velocity to 5 rps
D25ØØØ	Distance is 25,000 steps
G	Executes the move (Go)
VAR5=VAR5+1	Variable 5 counts up from 0
UNTIL(INXXXX111Ø_OR_VAR5>1Ø)	When the 111Ø input condition occurs, the programmable inputs or VAR5 > 10, the loop will stop
1"DONE_LOOPING	Quote command indicates that the loop is finished
1CR	Inserts a carriage return
1LF	Inserts a line feed
XT	Ends definition of sequence #10

Step ②

Use the Trace mode to display the commands as they are run.

```
> 1XTR1
> XR1Ø
```

The loop can be exited either by using the **DIN** command to satisfy the input state or letting the **VAR5** counter count to 10.

WHILE

With the **WHILE** command, all commands are repeated between **WHILE** and **NWHILE** until the **WHILE** condition is true. Enter the following sequence.

Step ①

Command	Description
> VAR5=Ø	Initializes variable 5 to 0
> 1XE1Ø	Erases sequence #10
> 1XD1Ø	Defines sequence #10
WHILE(INXXX111Ø_OR_VAR5<1Ø)	While input pattern is XXX111Ø or variable 5 is<10, repeat loop
A5Ø	Acceleration is 50 rps^2
AD5Ø	Deceleration is 50 rps^2
V5	Velocity is 5 rps
D25ØØØ	Distance is 25000 steps
G	Executes the move (Go)
VAR5=VAR5+1	Variable 5 counts up from 0
NWHILE	
1"DONE_LOOPING	Indicates that the loop is done
1CR	Inserts a carriage return
1LF	Inserts a line feed
XT	Ends definition of sequence #10

Step ②

Use the Trace mode to display the commands as they are run.

```
> 1XTR1
> XR1Ø
```

You can exit the loop with the **DIN** command (the input state does not match the **IN** command). You can also exit the loop by letting the **VAR5** counter get to 10. If the input pattern is not **XXX111Ø**, the loop will not be run.

Conditional Branching

You can use the **IF** statement for conditional branching. All commands between **IF** and **ELSE** are executed if the condition is true. If the condition is false, the commands between **ELSE** and **NIF** are executed. **ELSE** may not be required. The commands between **IF** and **NIF** are executed if the condition is true. Examples of these statements are provided.

☛ Note

If statements are evaluated at the time the command is executed they are not continually evaluated in the background.

- ❑ Error Flag
- ❑ User Flag
- ❑ Input State

Error Flag

The **IF** command checks to see if any error condition exists to execute a conditional command. This command is useful if you wish to trap different error conditions (Drive Disabled, User Fault Input Activated, Excessive Position Error, etc). Refer to the ***SX Software Reference Guide.***

Command	Description
> XE1Ø	Erases sequence #10
> XD1Ø	Defines sequence #10—if a fault occurs, sequence #10 will execute—defined with Set Fault or Kill Sequence (**XFK1Ø**) command
IF(ER1)	If hardware CCW limit switch is reached, run the following commands:
1"CCW_LIMIT_HIT	Display the error message
NIF	Ends IF statement
IF(ERX1)	If hardware CW limit switch is reached, perform the following commands:
1"CW_LIMIT_HIT	Display the error message
NIF	Ends IF statement
XT	Ends sequence
> XFK1Ø	Sets sequence #10 as the Fault sequence

User Flag

This example uses the pattern set by the User Flag (**SFL**) command to run the conditional commands. This command is useful if you wish to make a decision based on previous sequence executions that will set or clear the user flag bits. For example, if an application has several sequences, you can assign different bit patterns with the **SFL** command at the end of each sequence. If you select these sequences from the host computer, you may wish to make different moves depending on the sequence you ran. Refer to the ***SX Software Reference Guide*** for a detailed description.

Command	Description
> PS	Waits for the SX to receive a **C** command before executing the next command
SFL1Ø1Ø	Sets user flag bits 7 and 5 and clears bits 6 and 4, remaining bits are not altered
IF(FL1Ø1Ø)	If user flag bits 5 & 7 are set, and bits 6 & 4 are clear, perform these commands
A1Ø	Sets acceleration to 10 rps^2
V5	Sets velocity to 5 rps
D25ØØØ	Sets distance to 25,000 steps
G	Executes the move (Go)
NIF	Ends **IF** statement
C	Continues execution

If the **FL** pattern matches the **SFL** setting, the motor moves 25,000 steps. You can change the **SFL** pattern at different points in sequences to map a path for sequence execution.

Input State

The (IN) command compares the input pattern (**CW**, **CCW**, **HOME**, **REG**, and **I1** - **I8**) to execute the conditional commands. This command is useful for branching and performing conditional moves using the programmable inputs. For a detailed description of this command, refer to the ***SX Software Reference Guide***.

Command	Description
> 1XE5	Erases sequence #5
> 1XD5	Defines sequence #5
IF(INXXXX1Ø)	If **I1** is active and **I2** is not active, issue the following commands:
A1Ø	Sets acceleration to 10 rps^2
V5	Sets velocity to 5 rps
D25ØØØ	Sets distance to 25,000 steps
G	Executes the move (Go)
NIF	Ends IF statement
IF(INXXXXØ1)	If **I1** is inactive (open) & **I2** is active (closed), run the following commands:
A1Ø	Sets acceleration to 10 rps^2
V5	Sets velocity to 5 rps
D-5ØØØ	Sets distance to 5,000 steps in the opposite direction
G	Executes the move (Go)
NIF	Ends IF statement
IF(INXXXX1)	If **I1** is active, do the following command.
1"DONE	Ends message saying done
NIF	Ends **IF** statement
1XT	Ends sequence definition

Use the **DIN** command or the inputs themselves to execute the different trigger input states. You can use the Trace mode to see what commands are executed. The input state represents the state of all inputs regardless of whether they are dedicated limits or programmable inputs. This is different than the trigger (**TR**) command where only the inputs defined as triggers are used in the command.

Branching Using Variables and Boolean Logic

You can use the **IF** statement to branch based on variable values. Multiple comparisons can be made in one condition statement using the Boolean, **OR**, and **AND** functions as long as the statement doesn't exceed 40 characters.

Command	Description
> XE8	Erases sequence 8
> XD8	Defines sequence 8
VAR5=15	Variable 5 = 15
IF(VAR5<1Ø_AND_VAR4=2Ø)	If variable 5 < 10 & variable 4 = 20, run commands up to the **ELSE** command
A1ØØ	Sets acceleration to 100 rps^2
AD1ØØ	Sets deceleration to 100 rps^2
V5	Sets velocity to 5 rps
D25ØØØ	Sets distance to 25,000 steps
G	Executes the move (Go)
VAR5=VAR5-1	Variable 5 decrements one
ELSE	Ends IF statement
A1ØØ	Sets acceleration to 100 rps^2
AD1ØØ	Sets deceleration to 100 rps^2
V5	Sets velocity to 5 rps
D-5ØØØ	Sets distance to 5,000 steps in the CCW direction
G	Executes the move (Go)
NIF	Ends the IF statement
1XT	Ends the sequence definition

Motion Profiling Mode—On-the-Fly Changes

Motion Profiling mode allows you to execute buffered commands while a move is being made (on-the-fly). When you enter this mode, the SX will execute commands while the move profile is in progress. You can enter and exit this mode from within a sequence. This mode allows you to change velocity on-the-fly based on distance, turn on outputs based on distance, perform math and other commands while in motion. Changing the acceleration, deceleration, or distance parameters will not affect any move already in progress, but will be in effect for any subsequent moves. The following commands are used with Motion Profiling mode.

- **MPP**—Enter Motion Profiling Mode
- **NG**—Exits Motion Profiling Mode
- **DP**—Sets Distance Points within Motion Profiling Mode

While the SX is in Motion Profiling mode, you can execute most command while a move is being made. The exceptions are other move commands such as **G** or **GH**. When the SX reaches an **MPP** command, all subsequent commands will be executed until the **NG** command is encountered. An example of the **MPP** command is provided below.

```
XD1 D50000 V1 MPP G O1 TR1X1 V4 NG XT
```

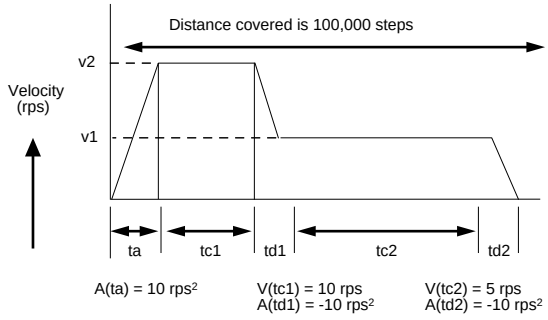
In this example, a 50,000-step move is made. The initial velocity is 1 rps. Motion begins with the **G** command. Output #1 is turned on with the **O1** command. The SX then runs the trigger command (**TR**) until the condition is true, at which point, it changes the velocity. When the SX encounters the **NG** command, it will not receive any additional commands until the 50,000-step move is completed. The 50,000-step move will be completed even if the **TR1X1** is not satisfied. The SX will however, wait until the **TR** condition is satisfied, at which point it will update the last specified velocity from **V1** to **V4**.

Changes Based on Distance

Changes are made based on distance using the distance point (**DP**) command. This command causes a delay in the processing of the commands until the motor reaches the specified distance point. Processing will continue once the distance point is reached. In this way, velocity changes and the activation of outputs can be based on distance.

The distance point is interpreted differently for Absolute mode versus Incremental mode. To change velocity on-the-fly based on distance, the Motion Profiling Mode (**MPP**) command must be used with the Distance Point (**DP**) command. The following sequence example executes the profile shown in the figure below.

```
1XD1 PZ D100000 V2 MPP G DP25000 O1 DP50000 V1 O0 NG XT
```



Motion Profiling Mode Example

In Incremental mode, commands are processed until the **DP** command is reached. The SX pauses at **DP** until the motor moves 25,000 steps. The SX then turns on output #1. The 25,000 steps are counted from the point at which **DP** is encountered. When **DP50000** is reached, the SX pauses until the motor moves an additional 50,000 steps. The SX then turns output #1 off and decreases the velocity to 1 rps. In this example, if the SX is in Incremental mode, the output will be turned on at 25,000 steps and turned off after the motor has moved a total of 75,000 steps from the beginning of the first move.

In Absolute mode, the value specified with **DP** is interpreted as an absolute position relative to the zero point location. In this example, the output would be turned on at 25,000 steps and turned off after the motor had passed the 50,000-step point.

Stopping Motion with a Stop Input

A common use of the Motion Profiling mode is to perform a continuous move and stop the move from the inputs. This can be done in two ways. You can define an input as a stop input. Motion can be stopped by activating the stop input.

The other method is to use the **STOP** command and place it within a sequence. The following example illustrates the steps of these two methods.

Step ① Define the input as a stop input.

Command	Description
> IN1D	Defines input #1 as a stop input

Step ② Create the sequence with a continuous move.

Command	Description
> XE1	Erase previous sequence #1
> XD1	Begins definition of sequence #1
V5	Sets velocity to 5 rps
A1ØØ	Sets acceleration to 100 rps^2
MC	Sets the SX to Continuous mode
MPP	Sets the SX to Motion Profiling mode
G	Executes the move (Go)
XT	Ends sequence #1 definition

Step ③ Execute sequence #1 (**XR1**). The motor will move at 5 rps and will not stop until you activate the stop input (input #1). Activate the stop input.

The motor will stop at a controlled deceleration. In this case, the buffer will be dumped and the sequences will not be finished. The SX can also be set to stop only the motion when the stop input is activated. Enabling the **SSH1** command will cause the stop input to stop the move in progress and whatever command is currently being executed and go onto the next command in the sequence or input buffer.

Step ④ Issue Display Parameters (**1DR**) command. The SX is still in Motion Profiling mode. Enter the **NG** command to exit the mode.

Stopping Motion with the STOP Command

The following step-by-step example illustrates the method of stopping a continuous move with the **STOP** command.

Step ① Configure input #1 as a trigger input.

Command	Description
> IN1A	Configures input #1 as a trigger input
> SSHØ	Disable **SSH** mode

Step ②

Create a sequence with a **STOP** command after the trigger.

Command	Description
> XE1	Erases previous sequence #1
> XD1	Begins the definition of sequence #1
V5	Sets velocity to 5 rps
A1ØØ	Sets acceleration to 100 rps^2
MC	Sets the SX to Continuous mode
MPP	Sets the SX to Motion Profiling mode
G	Executes the move (Go)
TR1	Activates trigger #1
STOP	Stops motion when the trigger condition is met
NG	Exits Motion Profiling mode
XT	Ends sequence #1 definition

Step ③

Issue the Display Parameters (**1DR**) command. The SX is still in Motion Profiling mode. In this example, an **NG** command was required after motion was stopped. When a **STOP** command is issued, the command buffer is emptied. Therefore, the commands that have not been executed in the sequence at the time the stop occurs will not be executed. In this example, the **NG** command is not executed because motion was stopped. In fact, **NG** can never be executed under these conditions. *If the* **STOP** *command is used to stop continuous motion, the* **NG** *must be issued either at the beginning of the next sequence, directly via the RS-232C interface or at some point in the sequence prior to the stop.*

To prevent the SX from stopping without finishing the sequence that it is currently executing, a software switch has been provided that will cause the SX to continue executing the sequence it was running when the **STOP** was issued. By entering the Clear/Save the Command Buffer on Stop (**SSH1**) command, the SX will stop motion when it encounters a **STOP** and continue processing the commands in the sequence. *Enter* **SSH1** *and repeat the previous step. Notice how the Motion Profiling mode is exited.*

Sequence Scan Mode and the Stop Command

In applications that use the Sequence Scan mode (see Sequence Select, Sequence Scanning, and Programmable Inputs and Outputs sections in this chapter) and the **STOP** command, you must use the (**SSH1**) command to prevent the Sequence Scan mode from being disabled. Under normal conditions, the Sequence Scan mode is aborted when the SX encounters a **STOP** command. If the **STOP** command is issued from a stop input or from within a sequence, the Sequence Scan mode will be aborted. In some cases, you will not want to abort the mode. **SSH1** allows the SX to complete the current sequence from the point at which the **STOP** was issued and continue in Sequence Scan mode. The **OSI1** command would allow continuing in Sequence Scan mode on a **STOP** but not continuing in the current sequence. The following example illustrates such a sequence.

Step ①

Configure input #1 as a trigger input and input #2 as a sequence-select input.

Command	Description
> IN1A	Configures input #1 as a trigger input
> IN2B	Configures input #2 as a sequence-select input

Step ②

Enable the Sequence Scan mode with the **SSJ1** command and the Clear/Save Command Buffer on Step mode **(SSH1**).

Step ③ Create a sequence with a **STOP** command after the trigger.

Command	Description
> XE1	Erase Previous Sequence #1
> XD1	Begins the definition of sequence #1
V5	Sets velocity to 5 rps
A1ØØ	Sets acceleration to100 rps^2
MC	Sets the SX to Continuous mode
MPP	Sets the SX to Motion Profiling mode
G	Executes the move (Go)
TR1	Activates trigger #1
STOP	Stops motion when the trigger condition is met
NG	Exits Motion Profiling mode
XT	Ends sequence #1 definition

Step ④ Execute the sequence by activating input #2. Stop the move at any time by activating input #1.

Step ⑤ The SX is still in Sequence Scan mode and the move can be repeated by activating Input #2 again (and stopped with Input #1). The SX is not in MPP mode in-between the sequence execution since the SX executed the **NG** command at the end of the sequence due to **SSH1**. **OSI1** would have saved the Sequence Select mode but not allowed executing the **NG** command after the **STOP**.

Other Uses of Motion Profiling Mode

Motion Profiling mode allows a great amount of flexibility in the complexity of the control of the SX during motion. You can turn on outputs, change velocity, and perform math functions. **The primary application concern to consider during sequence execution is the amount of time required to perform the commands.** In some cases, the execution of commands may depend on the motion. The following examples show additional uses of the Motion Profiling mode.

Turning on Inputs, Using Time Delays, and Math

Command	Description
> XE1	Erase previous Sequence #1
> 1XD1	Begins the definition of sequence #1
1PZ	Sets axis #1 position counter to Ø
1MC	Sets the SX to Continuous mode
1MPP	Sets SX to Motion Profiling mode
1G	Executes the move (Go)
1T.5	Sets a 0.5 second delay
1O11Ø	Turns outputs #1 and #2 on, #3 off
1T.5	Sets a 0.5 second delay
1OØØ1	Turns outputs #1 and #2 off, #3 on
REPEAT	Starts repeat loop
VAR1=VAR1+1	Increases variable #1 by 1
T.1	Sets a 0.1 second delay
UNTIL(POS>4ØØØØØ)	Continues looping until the SX's position is > 400,000
STOP	Halts command processing
NG	Exits SX from Motion Profiling mode
XT	Ends the definition of sequence #1

Triggers, input states (**INXX111**), time delays, and the distance points allow you to control when and where procedures occur during motion in your program. Motion Profiling mode offers you the flexibility to satisfy a variety of different application needs. In the above example, **SSH1** should be enabled to exit **MMP** (**NG**) after the **STOP** is executed.

Interfacing to the SX

This section discusses interfacing the SX to other equipment in a system using the programmable inputs and outputs.

- ❑ Input and Output Function Types
- ❑ Switches
- ❑ Sequence Selecting
- ❑ Thumbwheels
- ❑ Sequence Selecting from a Thumbwheel
- ❑ PLC Operation
- ❑ Sequence Selecting from a PLC
- ❑ Miscellaneous Control from a PLC
- ❑ RP240 Operator Panel

Programmable Inputs and Outputs

The SX has a very flexible input and output scheme for defining I/O in a way that is suitable for almost any application. There are 8 programmable inputs (**I1 - I8** on the front panel). The other four inputs are dedicated for limits, home, and registration. There are also 4 programmable outputs in addition to a dedicated Fault output. This section explains some of the functions that the inputs and outputs can perform and explains how to use thumbwheels for an interface with the SX. Using the inputs in combination with the outputs you can use up to 32 digits of thumbwheels with the SX. Refer to *Chapter 3, Installation* for more information on wiring the inputs and outputs to other equipment and later in this cchapter for wiring to the Compumotor TM8 Module.

Output Functions

You can turn the programmable outputs (**O1** - **O4**) on and off with the Output (**O**) and Immediate Output (**IO**) commands. Outputs **O1** - **O4** are factory set as programmable outputs. The Fault output is dedicated as a fault output. However, you can configure all of the programmable outputs to perform different functions (Moving/Not Moving, Amp Off, Strobe, etc.) with the Configure Output (**OUT**) command. Refer to the **OUT** command in the ***SX Software Reference Guide*** for descriptions of the available functions. You can use these outputs to turn on and off other devices (i.e., lights, switches, relays, etc.). The output functions have unique letter assignments.

A: Programmable Output
B: Moving/Not Moving
C: Sequence in Progress
D: At Soft or Hard Limits
E: At Position Zero
F: Fault Indicator
H: Shutdown Commanded
J: Strobe Out
K: Invalid Command Error
L: Position Error Fault
N: CW Software Limit Reached
P: CCW Software Limit Reached
R: CW Hardware Limit Reached
S: CCW Hardware Limit Reached
T: Output Based on Position
U: Pulse Output
Z: No Function Assigned

Command	Description
> PS	Pauses command execution until the SX receives a Continue (**C**) command
MN	Sets unit to Normal mode
LD3	Disables the SX's limits
A1Ø	Sets acceleration to 10 rps^2
V5	Sets velocity to 5 rps
D25ØØØ	Sets distance to 25,000 steps
OUT1A	Sets **O1** as a programmable output
OUT2A	Sets **O2** as a programmable output
OUT3B	Sets **O3** as a Moving/Not Moving output
O1Ø	Turns **O1** on and **O2** off
G	Executes the move
OØ1	Turn **O1** off and **O2** on
C	Initiates command execution to resume

This example defines **O1** and **O2** as programmable outputs and **O3** as a Moving/Not Moving output. Before the motor moves 25,000 steps, **O1** is turned on and **O2** is turned off. These outputs will remain in this state until the move is completed, then **O1** will turn off and **O2** will be turned on. While the motor is moving, **O3** remains on.

The active level of the programmable outputs can be changed with the **OUTL** command. Refer to the ***SX Software Reference Guide*** for more details.

Programmable Output	This output type gives the user on/off control over an output using the **O** or **IO** commands.
Moving/Not Moving Output	This output type indicates motion due to a **G** or **GH** command. This output does not indicate motion due to a position maintenance move.
Sequence in Progress Output	This output type indicates the SX is busy running a defined sequence.
At Soft or Hard Limit Output	This output type will indicate whenever any type of end of travel limit is hit during a move if the limits are enabled. This output is reset with the **ST0** or **ON** command.
At Position Zero Output	This output type indicates when the absolute position counter (**1PR** command) is equal to zero.
Fault Indicator	This output type indicates when certain fault conditions are true. The conditions that will activate it are:Over-temperature faultMotor faultLow-line fault (brown out)Excessive position error (cleared by **ST0** or **ON** command)Auto run mode activeUser fault input activeThese errors are only cleared by resetting the unit or cycling power unless otherwise noted.
Shutdown Commanded	This output type indicates a drive disabled condition caused by the **ST1** or **OFF** command. This output is reset with the **ST0** or **ON** command.
Strobe Output	This output type is used in conjunction with the SX's programmable inputs configured as data inputs. The strobe outputs will cycle through different output patterns to tell external devices which digit of information to put on the data inputs. Refer to later in this chapter for more information on this topic.
Invalid Command Error	This output type indicates that an invalid command was seen by the SX, either due to a value out of range, a syntax error, or an impossible request This output is reset by resetting or cycling power.
Position Error Fault	This output type indicates when the encoder input counter (**1PX**) differs from the number of pulses sent out (**1PR**) by more than the acceptable position error (set with the **CPE** command). The **DPE** command also reflects this positional error. This output is reset with the **ST0** or **ON** command.
CW Software Limit Hit	This output type indicates that the CW software limit was hit. This output is reset with the **ST0** or **ON** command.
CCW Software Limit Hit	This output type indicates that the CCW software limit was hit. This output is reset with the **ST0** or **ON** command.
CW Hardware Limit Hit	This output type indicates that the CW hardware limit was hit. This output is reset with the **ST0** or **ON** command.
CCW Hardware Limit Hit	This output type indicates that the CCW hardware limit was hit. This output is reset with the **ST0** or **ON** command.
Output Based on Position	This output type indicates that a certain position has been reached, either an incremental or absolute position, depending on the setting of the **OUTP** command. Refer to the **OUTP** command for more details and how to calculate the accuracy of this output.
Pulse Output	This output type generates a pulse train output that can be used for a limited motion axis output. The number of pulses sent out and the appropriate pulse width will be set with the **PUL** command. It is not intended as a full axis output and has no acceleration or deceleration ramp.
No Function Assigned	This output type has no function assigned to it and can be used to disable an output when needed.

Input Functions

The inputs can individually be programmed to perform any of the following functions. Each function has an assigned letter:

A:	Trigger	**M:**	Terminate Loop
B:	Sequence Select	**N:**	Data
C:	Kill	**P:**	Memory Lock
D:	Stop	**R:**	Reset
E:	Command Enable	**S:**	Go Home
F:	Pause/Continue	**T:**	Position Zero
G:	Go	**U:**	User Fault
H:	Direction	**V:**	Data Valid
I:	Synchronization	**W:**	Data Sign
J:	Jog+ (CW)	**X:**	Increase Following Ratio
K:	Jog- (CCW)	**Y:**	Decrease Following Ratio
L:	Jog Speed Select	**Z:**	No Function Assigned

The input functions are level sensitive unless otherwise specified.

Trigger Input

This input type is mainly used in conjunction with the **TR** command to pause command processing until the specified input pattern is satisfied. This input function is explained in more detail later in this chapter.

Sequence Select Input

This input type is used to remotely execute predefined sequences using the inputs. This input function is explained in more detail later in this chapter.

Kill Input

This input type is used to immediately halt all motion with no deceleration, stops the current sequence execution, and dumps the command buffer. Functions the same as the K command.

Stop Input

This input type is used to immediately stop the motor at the specified deceleration rate (**AD** command). It will also dump the sequence and command buffer unless the **SSH** or **SSL** command has been set previously.

Command Enable Input

This input type is used to enable or disable the drive. Functions the same as the **ON** and **OFF** commands. This input type is edge sensitive.

Pause/Continue Input

This input type is used to pause and continue command execution. This input will not pause motion in progress. The input is a pause in the active state and a continue in the inactive state. Functions the same as the **U** and **C** commands

Go Input

This input type is used to initiate a move. Functions the same as the **G** command. This input type is edge sensitive.

Direction Input

This input type is used to change the direction of the motor. The direction change will not affect a move in progress. Functions the same as the **H** command. This input type is edge sensitive.

Synchronization Input

This input type is used in the following self correction mode to adjust the following ratio. Refer to the **FSK** and **FSL** commands as well as *Chapter 5, SXF Following* for more information.

Jog CW Input

This input type is used to jog the motor in the CW direction. The jogging velocity is set with the **JVL** and **JVH** commands. Jogging is enabled with the **OSE** command. This input is normally level sensitive, except after stopping or killing a jog. In this case the input needs to see the edge transition again.

Jog CCW Input

This input type is used to jog the motor in the CCW direction. The jogging velocity is set with the **JVL** and **JVH** commands. Jogging is enabled with the **OSE** command. This input is normally level sensitive, except after stopping or killing a jog. In this case the input needs to see the edge transition again.

Jog Speed Select Input

This input type is used to select between the **JVH** and **JVL** velocities. If not defined or used, the jogging velocity defaults to **JVL**. This input will affect a jog in progress.

Terminate Loop Input

This input type is used to terminate an **L**, **N** command loop at the end of the current iteration. It will continue program execution with the statement immediately after the **N** command. This input type is edge sensitive.

Data Input	This input type is used to load parallel bytes of data into the SX. The data is put on the SX inputs in BCD format. This input type is explained in more detail later in this chapter.
Memory Lock Input	This input type is used to lock out sequence editing commands and a couple others. These commands are **XE**, **XD**, **RIFS**, **CPE**, **CPG**, and **CPM**. The SX will report back what they are currently set to but will not allow them to be changed while the memory lock input is active.
Reset Input	This input type is used to execute a software reset of the SX. Upon recovery, the power-up sequence will execute, unless a serious fault condition still exists. This input is equivalent to the **Z** command. This input type is edge sensitive.
Point Zero Input	This input type is used to zero the absolute position counter. It is equivalent to the **PZ** command. This input type is edge sensitive.
User Fault Input	This input type is used to tell the SX that a fault condition exists external to the Indexer. It may come from a pushbutton, PLC, or other device. This fault condition is latched and must be cleared by cycling power, resetting the SX, or issuing the **ON** or **STØ** command. This input type is edge sensitive.
Data Valid Input	This input type is used in conjunction with data inputs when loading data from an external source. If an input is defined as data valid, the SX will not load information with one of the data entry commands (**DRD**, **VRD**, **LRD**, **FRD**, **TRD**, **VARD**, and **XRD**) unless the data valid input is active. A data read is attempted as often as the **STR** command setting allows. If no inputs are defined as data valid, the data is read in synchronously according to the **STR** command. This input function is explained in more detail later in this chapter.
Data Sign Input	This input type is used in conjunction with data inputs to indicate to the SX whether the data is a positive or negative value. This input is evaluated at the end of the data read and an active input will give a negative sign. If no input is defined as a data sign input the sign will default to positive.
Increase Following Ratio Input	This input type is used to increase the following ratio on the fly when following is enabled and a move is in progress. This input will increase the following ratio by the set **FIN** value every 1 msec that the input is seen as active.
Decrease Following Ratio Input	This input type is used to decrease the following ratio on the fly when following is enabled and a move is in progress. This input will decrease the following ratio by the set **FIN** value every 1 msec that the input is seen as active.
No Function Assigned	This input type is used to disable the input from any functions. It can be used to temporarily prevent an input from executing any functions even though the input is still being activated externally. It will remain this type until redefined.

To designate each input to a particular function, use the Set Input Functions (**IN**) command. To see what the inputs are currently defined as, type **1IN**. To see the inputs' states, use the **1IS** command. Enter the following commands as an example.

```
> 1IN
```

Change input 1 to be a stop input by entering: **> IN1D**

Check that it was assigned properly by again entering: **> 1IN1**

With this method, you can assign all the inputs to any of the input functions listed above.

Switches

This section contains information on SX triggers and sequence scanning with inputs. Refer to *Chapter 3, Installation* for more information on wiring the inputs to other equipment.

Triggers

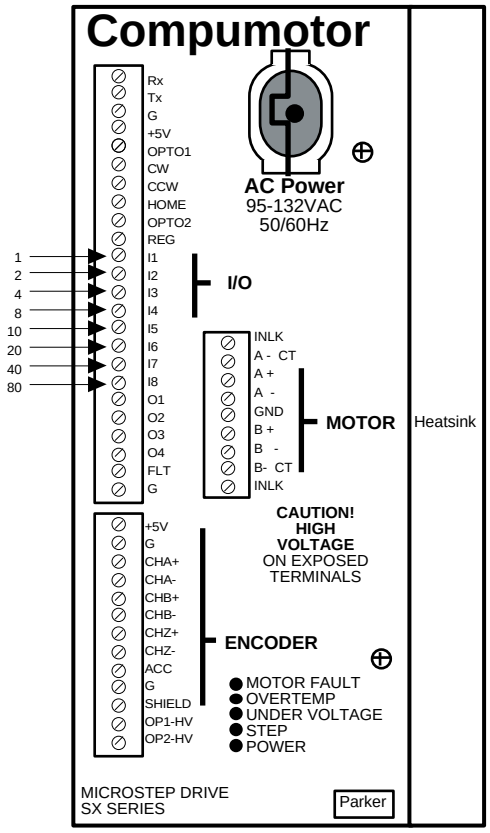
You can use the Wait for Trigger (**TR**) command to pause a sequence of buffered commands until one or more inputs reach a preferred state. Inputs **I1** - **I8** are set (default setting) to function as trigger inputs.

Command	Description
> IN1D	Sets **I1** as Stop input
> IN2A	Sets **I2** as Trigger # 1
> IN3A	Sets **I3** as Trigger # 2
> IN4A	Sets **I4** as Trigger # 3
> MC	Sets the unit to Continuous mode
> MPP	Enters the Motion Profiling mode
> V1Ø	Sets velocity to 10 rps
> A15	Sets acceleration to 15 rps^2
> AD15	Sets deceleration to 15 rps^2
> TR1	Waits for trigger input 1 (**I2**)to be on
G	Executes a go (Go) command
L	Loops infinitely
V5	Sets velocity to 5 rps
TRXØ1	Waits for trigger input 2 (**I3**) to be off and trigger input 3 (**I4**) to be on
V1	Sets velocity to 5 rps
TRX1Ø	Waits for trigger input 2 (**I3**) to be on and trigger input 3 (**I4**) to be off
N	Ends the loop

This example program configures **I1** as a stop input and **I2**, **I3**, and **I4** as trigger inputs. The command execution will pause (be buffered) until the first **TR** condition is satisfied by activating Trigger #1 (**IN2**). Because the SX is in Motion Profiling mode, it will execute the loop and subsequent commands during the move. As it reaches each trigger statement, it waits for that input state to become true and then executes the commands following each **TR** command. The loop is infinite so it will continuously toggle between 1 rps and 5 rps as the trigger statements come true and will not stop until the stop input is activated. If you activate **I1** during the operation of the SX, the Indexer immediately decelerates the motor at 15 rps^2 and clears the command buffer.

Sequence Select & Sequence Scanning

Inputs can be defined as sequence-select inputs. This allows you to execute sequences defined via RS-232C and stored in the SX's memory, by activating the sequence-select inputs. Sequence-select inputs are assigned BCD (binary coded decimal) weightings. **The lowest input number assigned as a sequence-select input will have the least significant value.** The following figure shows the BCD weights of the SX's inputs when all 8 inputs are configured as sequence-select inputs.



BCD Weight of SX Inputs

The following table illustrates **one possible** input configuration and binary weighting.

Input	Function	BCD Weighting
I1	Sequence Select	1
I2	Trigger	—
I3	Stop	—
I4	Sequence Select	2
I5	Sequence Select	4
I6	Sequence Select	8
I7	Sequence Select	10
18	Sequence Select	20

Input Configuration/BCD Weighting Example

The **IN** command is used to configure an input as a sequence select input. For example, **IN1B** defines Input #1 as a sequence select input. **IN5B** defines input #5 as a sequence select input.

If the inputs are configured as in the following table (Inputs 1-8 all defined as sequence select inputs), Sequence #6 will be executed by activating Inputs #2 and #3. Sequence #19 will be executed by activating Inputs #1, #4, and #5.

Input	BCD Weight
1	1
2	2
3	4
4	8
5	10
6	20
7	40
8	80

BCD Weighting of Sequence Select Inputs

To execute sequences, the SX must be in Sequence Scan mode. In this mode, the SX will continuously scan the input lines and execute the sequence selected by the active sequence-select lines. The **SSJ** command is used to enable/disable the Sequence Scan mode. When **SSJ1** is entered, the Sequence Scan mode is enabled. To disable the mode, enter **SSJØ**. The sequence select inputs are not latched and are only looked at for sequence scanning when no other sequence is being executed.

Once enabled (**SSJ1**), the SX will run the sequence number that the active sequence-select inputs and their respective BCD weightings represent. After executing and completing the selected sequence, the SX will scan the inputs again and run the selected sequence. If a sequence is selected that has not been defined via RS-232C, no sequence will be executed.

If it is not desirable for the SX to immediately execute another sequence after running the currently selected sequence, the Sequence Interrupted Run mode (**XQ1**) can be enabled. In this mode, after executing a sequence, all sequence-select lines must be placed in an inactive state before a new sequence can be selected. The active state of the inputs is determined by the **INL** command.

The Scan (**SN**) command determines how long the sequence-select inputs must be maintained before the SXexecutes the program. This delay is referred to as **debounce time**.

The **SN** value also determines how long the inputs must remain inactive if in **XQ1** mode. *Increasing the* **SN** *value can help with bouncy switches and electrically noisy environments.* The **SN** value can also help when running higher sequence numbers and combinations of inputs need to be synchronized. The higher the **SN** value, the more time allowed for all of the inputs desired to be activated.

Step ①

The following example demonstrates how to use Sequence Scan mode with Sequence Interrupted mode and the **SN** command.

Define a power-up sequence. (Sequence #100 is always the power-up sequence.)

Command	Definition
> XE1ØØ	Erases sequence #100
> XD1ØØ	Defines sequence #100
SSJ1	Enables Sequence Scan mode
SN2Ø	Sets scan time to 20 msec
XQ1	Sets SX to Interrupted Run mode
A1Ø	Sets acceleration to 10 rps^2
AD1Ø	Sets deceleration to 10 rps^2
V2	Sets velocity to 2 rps
IN1B	Sets Input 1 as a sequence-select input
IN2B	Sets Input 2 as a sequence-select input
IN3B	Sets Input 3 as a sequence-select input
IN4B	Sets Input 4 as a sequence-select input
IN5B	Sets Input 5 as a sequence-select input
IN6B	Sets Input 6 as a sequence-select input
IN7B	Sets Input 7 as a sequence-select input
IN8B	Sets Input 8 as a sequence-select input
OUT1C	Sets Output 1 as sequence-in-progress output
LD3	Disables the limits
XT	Ends the sequence definition

Every time you power up the SX Indexer, it executes Sequence #100 and enables the SX to read up to 100 sequences from the sequence-select inputs.

Step ②

Define any sequences that your application may require.

Command	Description
> XE1	Erases sequence #1
> XD1	Defines sequence #1
D2000	Sets distance to 2,000 steps
G	Executes the move (Go)
XT	Ends sequence #1 definition

Command	Description
> XE2	Erases sequence #2
> XD2	Defines sequence #2
D4000	Sets distance to 4,000 steps
G	Executes the move (Go)
XT	Ends sequence #2 definition

Command	Description
> XE3	Erases sequence #3
> XD3	Defines sequence #3
D8000	Sets distance to 8,000 steps
G	Executes the move (Go)
XT	Ends sequence #3 definition

Command	Description
> XE99	Erases sequence #99
> XD99	Defines sequence #99
D-14000	Sets distance to -14,000 steps
G	Executes the move (Go)
XT	Ends sequence #99 definition

Step ③

Verify that your programs were stored properly by uploading each entered sequence using the **XU** command (**1XU1 1XU2** etc.). If you receive responses that differ from what you programmed, re-enter those sequences.

Step ④

Run each program from the RS-232C interface with the Run Sequence (**XR**) command (**XR1**, **XR2**, etc.). Make sure that the motor moves the distance that you specify.

Step ⑤

Wire normally open switches to the inputs. Refer to *Chapter 3, Installation* for more information on wiring the inputs to other equipment.

Step ⑥

To execute sequences, cycle power to the SX. The system will execute sequence #100.

Step ⑦

You can now execute sequences by closing the corresponding switch or combination of switches.

- ❏ Close switch 1 to execute sequence #1
- ❏ Close switch 2 to execute sequence #2
- ❏ Close switches 1 & 2 to execute sequence #3
- ❏ Close switches 1, 4, 5, and 8 to execute sequence #99

Thumbwheel Interface

With the SX, you can use up to 16 digits of thumbwheels. The SX uses a multiplexed BCD input scheme to read thumbwheel data. Therefore, a decode circuit must be used for thumbwheels. Compumotor recommends that you purchase Compumotor's TM8 Module if you want to use a thumbwheel interface. ***The following section assumes that you are using Compumotor's TM8 module with the SX***.

Reading Parallel Data

The SX has seven parallel read commands that allow data to be read on inputs defined as Data Inputs. These commands are listed below:

Command	Description
> DRD	Read distance via thumbwheels
> VRD	Read velocity via thumbwheels
> LRD	Read loop count via thumbwheels
> TRD	Read time delay via thumbwheels
> VARDn	Read variables via thumbwheels
> XRD	Read sequence count via thumbwheels
> FRD	Read following ratio via thumbwheels (SX-F only)

The following section describes the method used to read parallel data and the different modes available (for a description of the individual commands, see the ***SX Software Reference Guide***). The software description of the parallel read commands assumes the SX is in **TW1** mode. The parallel read commands can be used in one of three modes selected by the **TW** command. A description of each mode is given below.

TWØ **TW0** mode reads two BCD digits at a time over the inputs. It requires that eight inputs be defined as Data Inputs, and at least one output be defined as a Strobe Output. Inputs 5-8 have a higher significance than inputs 1-4 in **TW0** mode. The number of times it reads a pair of BCD digits on the inputs is equal to the number of outputs defined as Strobe Outputs. The table below shows the output patterns during the strobing when all four outputs are Strobe Outputs. If fewer than four are used as Strobe Outputs, disregard the extra columns and rows in the table. (i.e. 2 outputs are Strobe Outputs Refer to the first two columns and rows in the table). The time delay between the changes are specified by the Strobe Output Delay (**STR**) command.

During each strobe time, the proper BCD data should be put on the SX's Data Inputs. If only one digit of information is desired, only four inputs need to be Data Inputs and only one Output as a Strobe Output. In this configuration, another input could then be defined as a Data Valid Input to trigger the data read instead of the **STR** value.

Strobe Out #1	Strobe Out #2	Strobe Out #3	Strobe Out #4
0	1	1	1
1	0	1	1
1	1	0	1
1	1	1	0

Ø = Inactive, 1 = Active

TW1 **TW1** mode is compatible with Compumotor's Thumbwheel Interface Module (TM8). It reads one BCD digit at a time over the data inputs. Four of the eight inputs must be defined as Data Inputs with the lowest numbered input having the least significance. Three of the four outputs must be defined as Strobe Outputs.

When a parallel read command is issued, the Strobe Outputs toggle in a binary pattern as shown in the table below. One digit is read in from the four lowest numbered Data Inputs for each binary pattern toggled on the outputs. The Strobe Outputs remain in each state for the amount of time specified by the Strobe Output Delay (**STR**) command and the inputs are read at the end of each strobe delay time. To allow for external or varying strobe rates, an input can be defined as a Data Valid Input. The SX will not read the inputs until it sees a data valid signal for each digit being read. In this case the **STR** value determines how long the data valid input must remain active to be seen.

TW1 mode also allows you to select a range of digits to be read and to provide a scale factor for the data. Three values can be added to the end of each command such as **DRDcde**. Variables c and d select the range of digits to be read from the inputs and may range from 0-7 to represent the desired thumbwheel digits. The TM8 Module's left most digit is 0 and the right most is 7. The variables c and d must satisfy the equation 0 c d 7. The variable e scales the thumbwheel read value by 10^e. If any of the extra digits are to be used, all three must be specified.

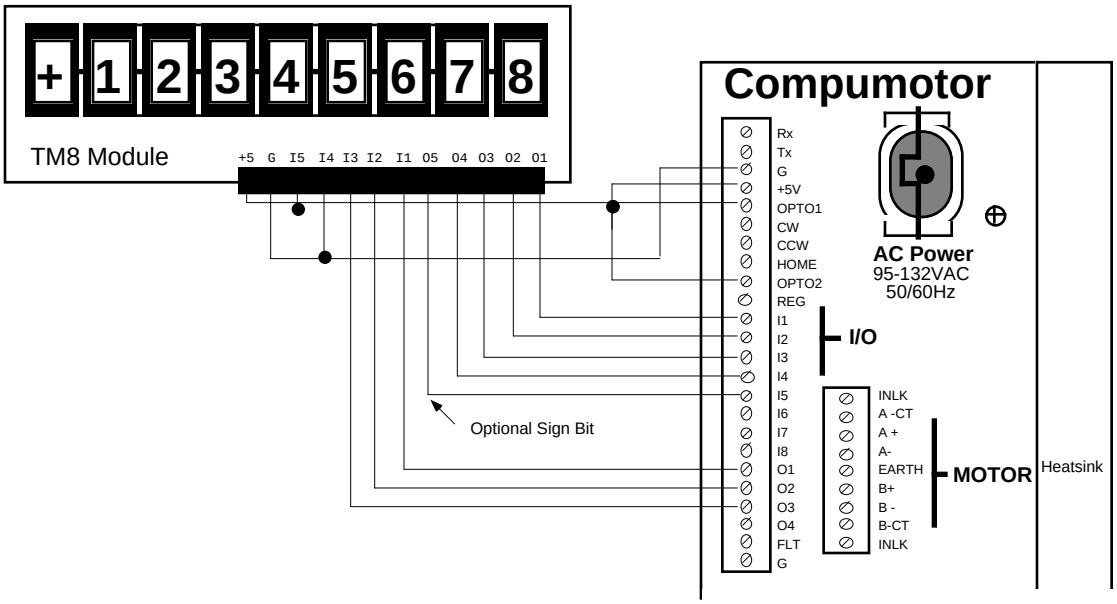
Strobe #1	Strobe #2	Strobe #3
1	1	1
0	1	1
1	0	1
0	0	1
1	1	0
0	1	0
1	0	0
0	0	0

Ø = Inactive, 1 = Active

TW2 **TW2** mode is identical to **TWØ** except that only one BCD digit is read at a time rather than two. Hence, only four inputs must be defined as Data Inputs and a maximum of four digits can be read, one for each output defined as a strobe output.

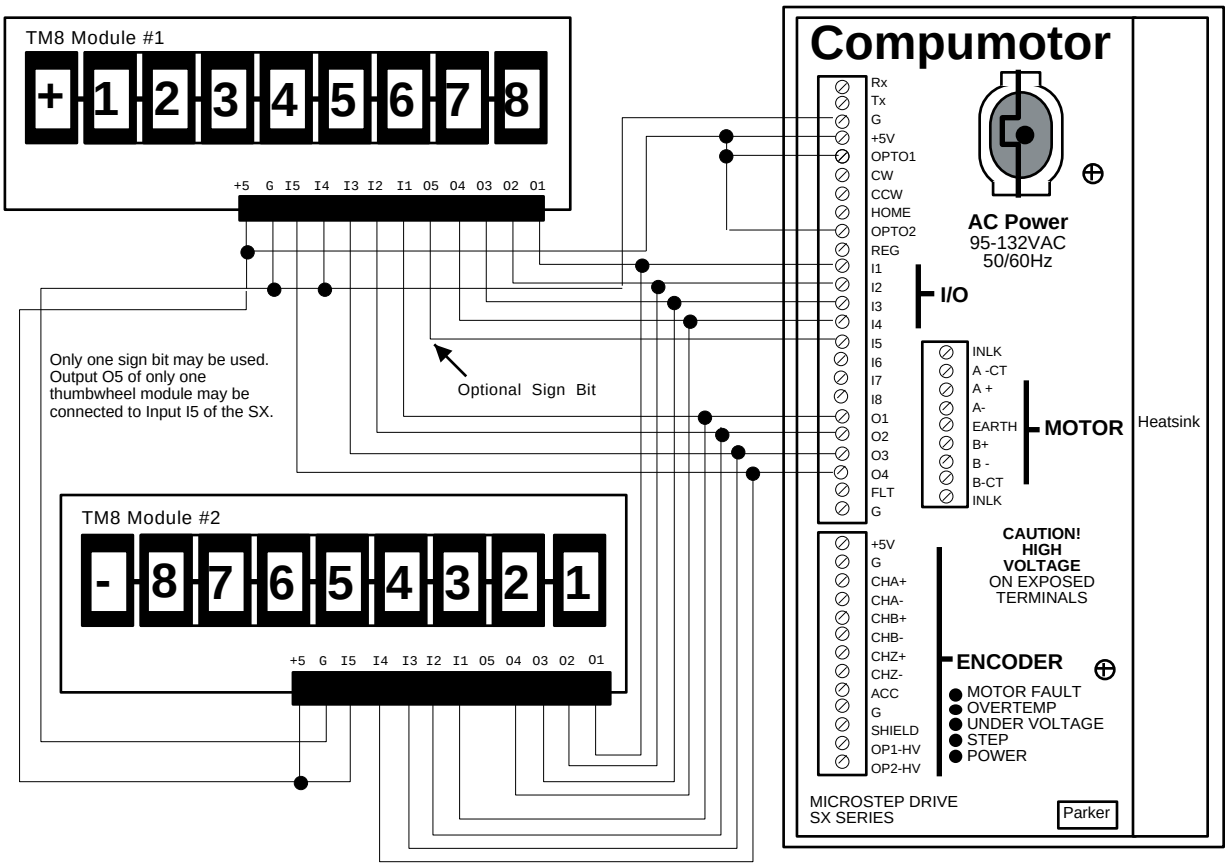
When reading digits from the Thumbwheel the most significant number (digit Ø) is read in first. The output strobe pattern is the same as for **TWØ** mode. Refer to the previous table for the outputs you are using. The following is an example using one or two TM8 Modules.

Step ① If you are using one TM8 Module, wire your module to the SX as shown in the first figure. If you are using two TM8 Modules (the maximum allowed per **each** SX), wire the modules as in the second figure.



Wiring 1 Thumbwheel Unit (TM8) to the SX

If you are powering an encoder with 5V from the SX along with one or two TM8 Modules, an external 5V supply must be used to power OPTO1 and OPTO2 or the encoder.



Wiring 2 Thumbwheel Units (TM8) to the SX

Step ②

Configure your SX as follows:

Command	Description
> OUT1J	**O1** configured as a strobe
> OUT2J	**O2** configured as a strobe
> OUT3J	**O3** configured as a strobe
> IN1N	**I1** configured as a data input
> IN2N	**I2** configured as a data input
> IN3N	**I3** configured as a data input
> IN4N	**I4** configured as a data input
> IN5W	**I5** configured as a sign input (optional)
> INLØ	Inputs configured active low
> STR5Ø	Data strobe time of 50 ms per digit read. If using one TM8 Module, you should now be ready to read in thumbwheel data. If using two TM8 Modules, enter the additional set-up commands. **Minimum recommended strobe time for the TM8 module is 10 ms.**
> OUT4A	**O4** configured as a programmable output
> OUTLØ	Outputs set active low
> O1	Set output 4 high—this enables TM8 Module #1

Step ③

Set the thumbwheel digits on your TM8 Module to **+12345678**. If using two TM8 Modules, set the second module to **-87654321**. To verify that you have wired your TM8 Module(s) correctly and configured your SX I/O properly, enter the following commands:

Command	Description
> DRD	Request distance data from all 8 thumbwheel digits
> 1D	Displays the distance read—**D+0012345678.** If you do not receive the response shown, return to step #1 and retry.

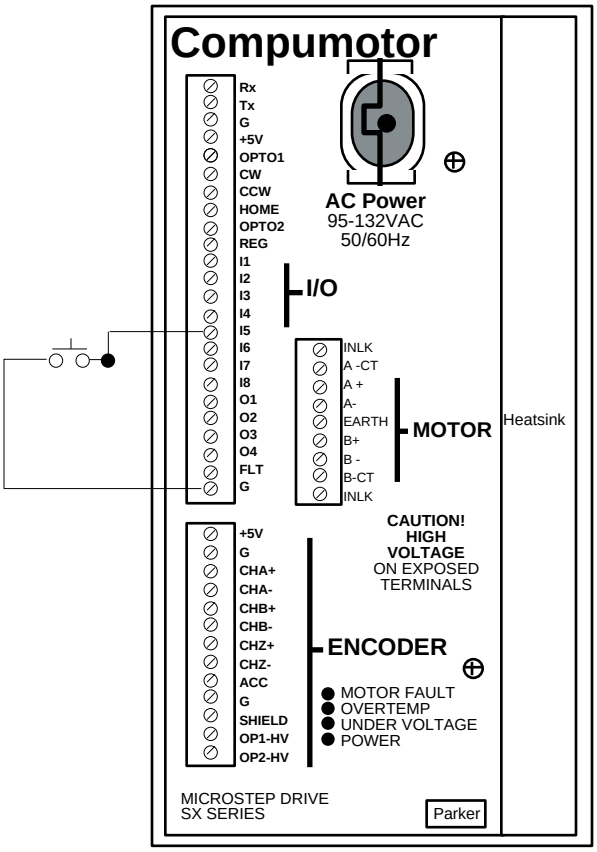
If you are using two TM8 Modules, enter the following commands:

Command	Description
> O0	O4 becomes 0V—this disables Module #1 and enables Module #2.
> DRD	Request distance data from all 8 thumbwheel digits
> 1D	Displays the distance read**–D-0087654321** *The sign is positive.*—only one sign digit may be used when two TM8 Modules are used. If you do not receive the response shown, return to step #1 and retry.
> 01	Re-enables the first TM8 Module.

Selecting Sequences with Thumbwheel Module

The following example shows how the SX is often used with thumbwheels. In the example, only three sequences are entered. As many as 100 sequences may be defined and up to 100 may be executed with the TM8 Module. Sequence #100 is automatically executed during power-up, reset, or by **XR100**. Refer to the Reset (**Z**) command in the ***SX Software Reference Guide***.

Ensure that the thumbwheel module is properly installed (as shown in the previous figures). Wire input 5 as shown in the following figure. The switch shown in this configuration is a data valid switch. Note that the sign bit is not being used.



Sequence Start Configuration

Step ①

Define a power-up sequence. Set inputs **I1** - **I4** as data inputs and **I5** as a data valid input. Enter the following program.

Command	Description
> XE1ØØ	Erases sequence #100
> XD1ØØ	Begins definition of sequence #100
IN1N	Sets I1 as a data input
IN2N	Sets I2 as a data input
IN3N	Sets I3 as a data input
IN4N	Sets I4 as a data input
IN5V	Sets I5 as a data valid input
INLØ	Sets ØV as active level
STR1Ø	Sets strobe time of 10 ms per digit
OUT1J	Sets O1 as a strobe output
OUT2J	Sets O2 as a strobe output
OUT3J	Sets O3 as a strobe output
OUTLØ	Outputs set active low
L	Start a continuous loop
XRD67Ø	Run the sequence displayed on thumbwheel digits 6 & 7
N	Ends loop
> XT	Ends definition of sequence #100

Step ②

Define any sequences that your application may need.

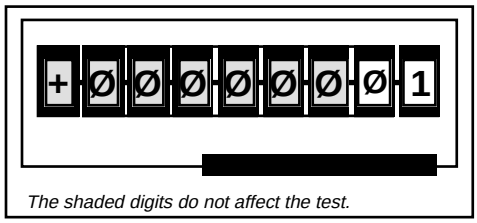
Command	Description
> XE1	Erases sequence #1
> XD1	Begins definition of sequence #1
MN	Sets to Normal mode
A25	Sets acceleration to 25 rps^2
AD25	Sets deceleration to 25 rps^2
V5	Sets velocity to 5 rps
D25ØØØ	Sets distance to 25000 steps
G	Executes the move (Go)
> XT	Ends definition of Sequence #1
> XE2	Erases sequence #2
> XD2	Begins definition of sequence #2
MN	Sets to Normal mode
A25	Sets acceleration to 25 rps^2
AD25	Sets deceleration to 25 rps^2
V5	Sets velocity to 5 rps
D1ØØØØ	Sets distance to 10000 steps
G	Executes the move (Go)
> XT	Ends definition of sequence #2
> XE99	Erases sequence #99
> XD99	Begins definition of sequence #99
MN	Sets to Normal mode
A25	Sets acceleration to 25 rps^2
AD25	Sets deceleration to 25 rps^2
V5	Sets velocity to 5 rps
D-35ØØØ	Sets distance to 1000 steps
G	Executes the move (Go)
> XT	Ends definition of sequence #99

Step ③ Reset the SX. This will execute the power-up sequence (Sequence #100).

Command	Description
> Z	Resets the SX

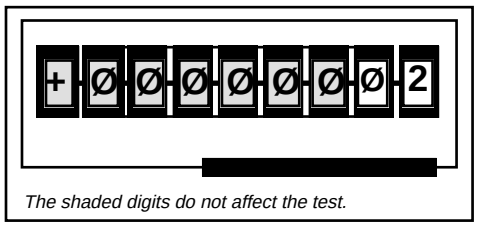
Step ④ Set thumbwheel digits 7 and 8 to **Ø1** and activate the Data Valid Input (#5) to move 25,000 steps CW.

(Execute Sequence #1)



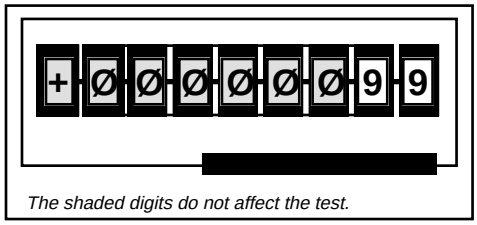
Step⑤ Set thumbwheel digits 7 and 8 to **Ø2** and activate the data valid input (#5) to move 10,000 steps CW.

(Executes Sequence #2)



Step ⑥ Set thumbwheel digits 7 and 8 to **99** and activate the data valid input (#5) to move 10,000 steps CCW.

(Executes Sequence #99)



If you select an invalid or unprogrammed sequence with the thumbwheels, no motion will occur and if the **SSN** command is enabled you will get the error message:

***UNDEFINED_SEQUENCE**

PLC Operation

This section explains and provides examples of how to use a PLC with the SX.

Interfacing with a PLC

In many applications, it is desirable to interface to a PLC. The SX performs the motion segment of a more involved process controlled by a PLC. In these applications, the PLC will start sequences, load data, manipulate inputs, and perform other specific input functions to control the SX and the motion segment of a process. This section assumes the SX is in **TW1** mode. If **TWØ** or **TW2** mode is desired, refer to the Thumbwheel section for more details.

As in the thumbwheel case, the PLC can be used to enter data for sequence-select (**XRD**), distance (**DRD**), velocity (**VRD**), loop count (**LRD**), time delay (**TRD**), variable data (**VARD**), and Following Ratio (**FRD**). Refer to the ***SX Software Reference Guide*** for more details on these commands.

To read data from the PLC, four of the SX's inputs must be configured as data inputs. If a sign digit is required, an input should be configured as a sign input. Configure 3 outputs as data strobe outputs. The active level of the inputs can be changed using the **INL** command. The active level of the outputs can be changed using the **OUTL** command.

When the SX executes a data read command, it will cycle its outputs and read BCD data as shown in the following table. The strobe outputs tell the PLC what digit of data to put on the SX inputs for each step in the process.

The Strobe Output Delay Time (**STR**) command sets the maximum rate that the SX will cycle through the output levels. The SX synchronously cycles through the states at the **STR** time if no data valid line is used. If a data valid line is used, the SX maintains its current state until the data valid input is activated. This allows the PLC to control the SX's data strobe rate.

The following table shows inputs 1-5 and outputs 1-3 being used. Any combination of inputs/outputs may be used. Their significance is always from the lowest numbered one to highest numbered one.

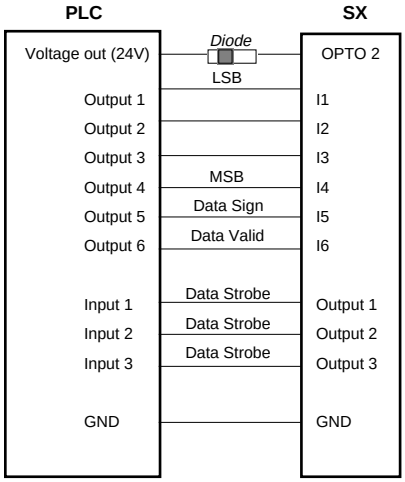
SX Outputs			Corresponding Digits Read	SX Inputs				
01	**02**	**03**	**Data In (Active Low)**	**I1**	**I2**	**I3**	**I4**	**I5**
1	1	1	MSD (Digit 1)	lsb			msb	± sign bit
0	1	1	Digit 2	"			"	"
1	0	1	Digit 3	"			"	"
0	0	1	Digit 4	"			"	"
1	1	0	Digit 5	"			"	"
0	1	0	Digit 6	"			"	"
1	0	0	Digit 7	"			"	"
0	0	0	LSD (Digit 8)	"			"	"

Ø = Inactive, 1 = Active (Connection to GND)

Data Strobing Table

Simply put, the SX executes a parallel read command and if there is no data valid input defined, places the first strobe pattern on its outputs and waits a time specified by the **STR** value. At the end of this time it reads the four lowest numbered Data Inputs, places the next strobe pattern on its outputs, and again waits for the **STR** value before reading its Data Inputs. This pattern repeats until all of the strobe patterns have been executed. If there is a Data Valid Input, the SX sees the Data Valid Input go active before each input read and strobe pattern change.

This figure shows a possible PLC-to-SX connection that would allow the SX to input data from the PLC. A data valid line controls the SX's strobe rate. A data sign line is also used.



PLC/SX Connnection

CAUTION

If voltages of 13VDC - 24VDC are used on OPTOs 1 & 2, a zener diode should be placed in series with the DC voltage supply and the OPTOs. If your PLC outputs are at a 13-24 Volt level a zener diode should be placed on each SX input with the same polarity as the OPTO2 diode shown in the previous figure. This will prevent reverse biasing and possibly damaging the SX inputs. Refer to Chapter 6, Hardware Reference for diode specifications.

If the SX executes a Read Distance Via Parallel I/O (**DRD**) command, the following events must occur to transfer distance data from the PLC to the SX. This is assuming **INL**, **OUTLØ**, and the same inputs/outputs as used in the previous figure.

① The SX executes a **DRD** command and places its outputs #1 - #3 at ØV.

② The PLC places and holds a BCD digit at the SX's inputs #1 - #4. (This value will be the most significant distance digit). The PLC places a sign value at SX input #5. (This sign bit **must be the same for each digit read**.)

③ When the data is valid, the PLC should activate the data valid line for (**STR**) milliseconds. The SX will read the digit and sign values.

④ The PLC must deactivate the data valid line.

⑤ After reading a data valid, the SX will place its output #1 in an inactive state and outputs #2 and #3 at ØV.

⑥ The PLC must place and hold its second BCD digit at the SX's inputs #1 - #4. The PLC should place the same sign value as that given for input #5.

⑦ When the data is valid, the PLC must activate the data valid line for (STR) milliseconds. The SX will read this second BCD digit as the second significant distance digit.

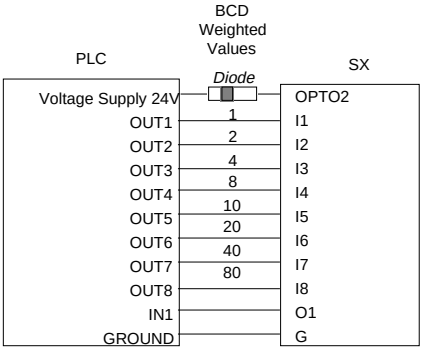
⑧ The PLC must deactivate the data valid line.

This process continues until the SX reads the eighth digit (LSD). At this point, the SX enters the eight digits read into its distance register and proceeds with the execution of subsequent commands.

Sequence Select With a PLC

The PLC can execute sequences through two different methods. First, the sequences may be selected by using the inputs defined as Sequence-Select Inputs. In this case, the input lines are BCD weighted (refer to the following figure). The SX must be in Sequence Scan mode (**SSJ1**) and may either operate in Interrupted mode (**XQ1**) or Continuous Scan mode (**XQØ**). Refer to the earlier switch and thumbwheel discussions for more information on Sequence Select methods.

The PLC activates the lines that will execute the desired sequence. It may be desirable to have the SX indicate to the PLC when it has completed a sequence or to indicate to the PLC when it should select another sequence. A programmable output can be used for this handshake to the PLC.



PLC Connection

CAUTION

If voltages of 13VDC - 24VDC are used on OPTOs 1 & 2, a zener diode should be placed in series with the DC voltage supply and the OPTOs. If your PLC outputs are at a 13-24 Volt level a zener diode should be placed on each SX input with the same polarity as the OPTO2 diode shown in the previous figure. This will prevent reverse biasing and possibly damage the SX inputs. Refer to Chapter 6, Hardware Reference for diode specifications.

Miscellaneous
Control by a PLC

You can use a PLC to control the activation of the inputs for many of the input functions that the SX supports (see the **Programmable Input** section). For example, you can use the PLC to stop, kill, go, go home, or reset the SX.

RP240 Operator Panel

Refer to the ***RP240 User Guide*** for information on using it with the SX.

Rotary vs. Linear Indexers

Most Compumotor Indexers are used for rotary motor systems. Hence, velocities and accelerations are selected in rps and rps^2 respectively. The default is often 25,000 steps per revolution. For linear motors, acceleration and velocities are usually defined in g's and inches per second (ips) respectively. Use the following equation to convert rps^2 to g's ($1g = 386\ ips^2$).

$$A[g] = \frac{A[rps^2] \bullet \text{Rotary Resolution [steps/rev]}}{\text{Linear Resolution [steps/in]} \bullet 386\ ips^2}$$

For example, if the rotary resolution is 25,000 steps/rev, the acceleration value is 100 rps^2, and the linear resolution is 10,000 steps/in. The equation is as follows:

$$\frac{100\ [rps^2] \bullet 25000\ [steps/rev]}{10000\ [steps/in] \bullet 386\ ips^2} = 0.648\ g$$

Use the following equation to convert rps to ips:

$$V[ips] = \frac{V[rps] \bullet \text{Rotary Resolution [steps/rev]}}{\text{Linear Resolution [steps/in]}}$$

For example, if the resolutions are the same as defined above, and the velocity value is 1 rps, the equation would be as follows:

$$\frac{1\ [rps] \bullet 25000\ [steps/rev]}{10000\ [steps/in]} = 2.5\ [ips]$$

☛ ***Helpful Hint:***

Rotary *vs* Linear Indexer Example

① Set the unit with the following move parameters:

- ❑ Acceleration = 1000 rps^2
- ❑ Velocity = 1 rps
- ❑ Distance = 10,000 steps

② Execute the **G** (Go) command:

If the resolution is 25,000 steps/rev, the forcer should move 1 inch at a velocity of 2.5 ips.