

Eggert Electronics Engineering

Daniel N. Eggert • 125 Pasa Por Aqui Ln. • Alamogordo, NM 88310 • Telephone (575) 437-0698

SSC1B Interface User's Manual

The SSC1B interface is a microstepping bipolar stepper motor controller that utilizes pulse width modulated current controlled motor drive output circuitry and a switchmode DC-DC converter for logic power. These features make the interface very efficient in power dissipation. The SSC1B interface circuit board controls one 2-phase bipolar stepper motor and 8 digital I/O lines with commands sent from a computers serial port. Up to 16 SSC1 system circuit boards can be connected to form a network that is controlled by commands sent from a single serial port. Stepper motor motion is performed by the SSC1B interface in the foreground while it receives commands in the background. You can send commands to read the stepper motor status word, or read the remaining step count while the stepper motor is in motion. There are 4 general-purpose digital inputs and 4 general-purpose digital outputs on each SSC1B system interface circuit board for a total of 64 digital inputs and 64 digital outputs in a network. The digital inputs have both latched and unlatched states that you can read with a system command. The input logic states of the 4 general-purpose digital inputs can be reversed at any time with a command so that a logic high input state can be represented by a logic low state. The 4 general-purpose digital inputs can also be individually programmed as a sense input to stop the stepper motor automatically when an input logic state changes. In addition, there are 2 digital inputs dedicated to sensing the stepper motors mechanical limits of travel. The Mechanical limit sense inputs can be reversed during initialization if needed. The 4 general-purpose digital output states are controlled by a system command. The digital outputs can be configured as either TTL level or high power transistor level outputs.

Figure 1 illustrates how the various components of an SSC1 system are interconnected to the SSC1B interface circuit board.

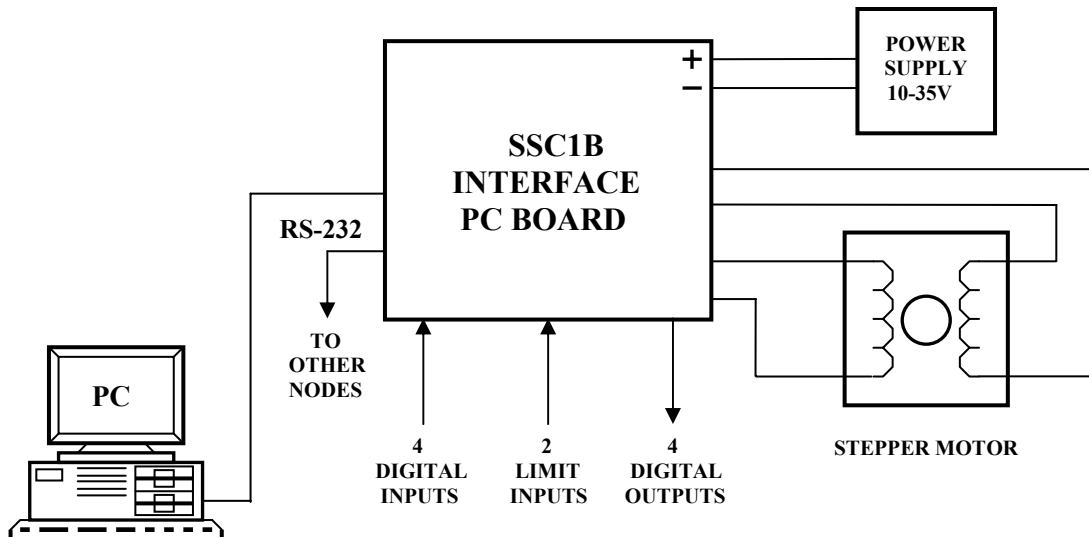


Figure 1. SSC1B Interface Interconnect

Copyright © 2004 Eggert Electronics Engineering. All rights reserved

February 2004

About the Interface

The SSC1B interface parts layout diagram is located at the back of this manual for reference. The interface incorporates a Microchip PIC16C63A embedded controller for command interpretation and control of an Allegro A3977SED dual full bridge microstepping driver IC. J1 is a 2-pin header on the circuit board that can be shorted momentarily to manually reset the interface circuit. Because of the wide input voltage range of the 5-volt switching regulator used for logic power, only one common power input is needed for the interface. The typical circuit board quiescent current is approximately 40 milliamps. LED D5 near the center of the circuit board is a power on indicator. The RS-232 serial port connections are made at terminal strips TS2 and TS3 near the bottom left edge of the interface circuit board. The SSC1B serial port parameters are 9600 BAUD, NO PARITY, 8 DATA BITS, and 1 STOP BIT. These parameters are fixed and cannot be changed. The "COM" LED D2 located near the upper center of the interface circuit board indicates serial communications activity. It will pulse on whenever there is any receive or transmit activity. The serial communications cable from the host PC, or from another SSC1 system circuit board on the network, is connected to the PORT1 RS-232 terminal strip TS2 pins TX, RX and common ground connection G. When two or more SSC1 interface circuit boards are configured as a network, the PORT2 RS-232 terminal strip TS3 pins TX, RX and common ground pin G are used to connect serial communications to the next interface circuit board in the network. Figure 4 illustrates how to connect multiple SSC1 system interface circuit boards (nodes) to form a network. The digital inputs are connected at terminal strip TS5 terminals 1 through 4. Their common ground connection is labeled G. A terminal labeled "+5V" supplies 5 volts DC to any external sensors or input conditioning circuitry that is required for your application. Caution should be used not to accidentally short this terminal to ground, or overload the interfaces on board 5-volt regulator when making connections to this terminal. There is approximately 250 milliamps of current available at this terminal for use with your external circuitry. All of the SSC1B interface digital inputs are pulled up to +5 volts by 10K ohm resistors. The mechanical limit sense inputs are connected at terminal strip TS1 pins CW and CCW with their common ground connection labeled G. The SSC1B digital outputs are present at terminal strip TS4 terminals 1 through 4 with their common ground connection labeled G. The digital outputs can be selected as either TTL level outputs or high-powered open drain transistor outputs with the jumper positions of J4 through J7. The shorting jumper is placed across the center pin and the left pin labeled "L" if a TTL logic output is required. The shorting jumper is placed across the center pin and the right pin labeled "T" if a high-powered open drain transistor output is required. Terminal strip TS4 has a terminal labeled "+" which can supply power to your external digital output devices such as LED's and relays etc. The voltage source for this terminal can be selected as either +5V or V+ (pc board power input) with the jumper position of J8 that is located to the right of TS4. Fuse F1 is rated at 5.0 amps and protects all of the interface circuitry including the V+ selection present at jumper J8. If you are going to sink high currents with the digital outputs then an alternative voltage source is recommended. The 4 digital output open drain power HEXFET transistor outputs are rated at 30 volts and over 20 amps of current. You can sink a constant current of 2.0 amps without any concern for overheating. You can sink higher currents intermittently, but the transistors have no heat sinks attached and caution should be taken to avoid overheating.

Motor and Power Supply Connections

The motor and its power supply connections are shown in figure 2. The power input for the SSC1B interface logic and stepper motor drive must be between 10 VDC and 35 VDC. The power supply connections are made at terminal strip TS7. Terminal strip TS7 is located at the upper right side of the interface circuit board. The interface power input circuitry includes an on board fuse and polarity reversal protection. The fuse will blow if the power input polarity is reversed and power is applied. The power supply voltage that is needed for your application depends on the high speed performance that is required

of the motor. Higher power supply voltages will yield better motor performance at higher speeds. The power supply input voltage should be between 3 and 6 times the motors rated voltage. The absolute maximum power supply input voltage is 35 VDC. Damage will occur to the interface circuitry if the input voltage exceeds 35 VDC.

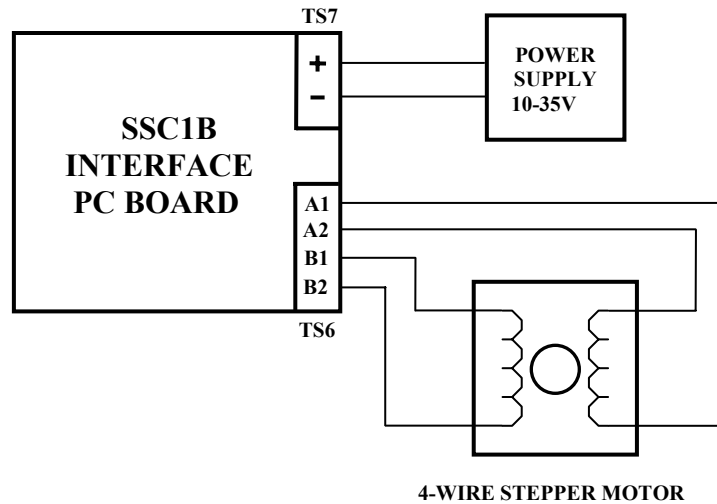


Figure 2. The SSC1B Interface Motor and Power Supply Connections

The motor connections are made at terminal strip TS6. This terminal strip is located at the center of the right edge of the interface circuit board. Any motors with 4-wires, 6-wires or 8-wires can be used with the SSC1B interface. Figure 2 above illustrates how a 4-wire motor is connected to the interface. 6-wire motors can be connected in either half winding or full winding configuration. A 6-wire motor that is connected in half winding configuration will perform the same as it would when used with a unipolar driver circuit. For half winding configuration of a 6-wire stepper motor, connect the coils center tap and one end of a winding leaving the other end of the winding disconnected as in Figure 3A. For full winding configuration of a 6-wire motor, connect the coil end wires leaving its center tap unconnected as in Figure 3B. Figures 3C and 3D illustrate how to connect 8-wire motors in either series or parallel configuration. Consult with the motor manufacturer's data sheets for the proper coil polarity and connections.

CAUTION! Always turn the power supply off when connecting or disconnecting the motor.

If the motor turns in the opposite direction that it is suppose to turn, remove power from the interface and reverse the connections of the wires connected to terminals A1 and A2 on terminal strip TS6. Reversing the connections of just one motor coil will reverse the direction of rotation.

CAUTION! The SSC1B interface will become damaged if the motor leads are shorted together.



Figure 3A. 6-Wire Motor Center to End Connection

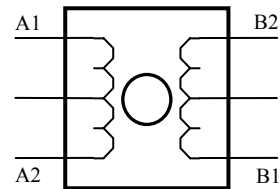


Figure 3B. 6-Wire Motor End to End Connection

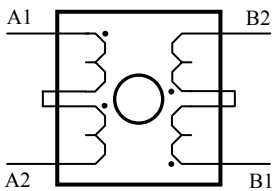


Figure 3C. 8-Wire motor Series Connection

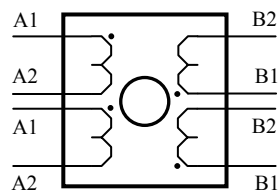


Figure 3D. 8-Wire Motor Parallel Connection

Setting The Driver Output Current

The specific driver output current required for your motor is selected by installing a resistor across the R1 and R2 terminals of terminal strip TS8. Terminal strip TS8 is located on the bottom edge of the circuit board near the right side. Table 1 below provides a list of motor currents in the left column along with the specific resistor value needed for the current selection next to it in the right column. The resistor values in the table are standard 1% values. You can use the closest 5% resistor value for applications where motor currents are not critical.

MOTOR AMPS	RESISTOR	MOTOR AMPS	RESISTOR
0.30	63.40K	0.80	11.30K
0.35	51.10K	0.85	9.53K
0.40	42.20K	0.90	7.87K
0.45	35.70K	0.95	6.34K
0.50	30.10K	1.00	4.99K
0.55	25.50K	1.05	3.83K
0.60	21.50K	1.10	2.74K
0.65	18.70K	1.15	1.74K
0.70	15.80K	1.20	845
0.75	13.30K	1.25	0 (wire jumper)

Table 1. Current Setting Resistor Values

Serial Interface To PC

The SSC1B interface communicates with the host PC using the fixed serial port parameters of 9600 BAUD, NO PARITY, 8 DATA BITS and 1 STOP BIT. The RS-232 serial interface connection from the SSC1B interface to the host PC uses a 3-wire cable. This cable can be up to 50 feet long. Shielded cable is recommended for long distances. If a shielded cable is used, the shield should be connected at connector pin 1. Do not connect the shield to anything at the SSC1 interface end of the cable. A 9-foot cable assembly with a 9-pin connector is supplied with the SSC1 system. There are 3 tinned wires at the end of the cable that connect to the PORT1 terminal strip TS2 on the SSC1B circuit board. Connect the black wire to the terminal labeled G. Connect the red wire to the terminal labeled RX. Connect the remaining green or white wire to the terminal labeled TX. Your PC will have either a 9 or 25 pin serial port connector. You may have to use a 9 to 25 pin adapter with the cable that is supplied with the SSC1 system. Use figure 3 as a guide when fabricating your own cable assembly. Be sure to include the 2 jumper wires on the connector. QuickBasic programs may not work properly without these jumpers.

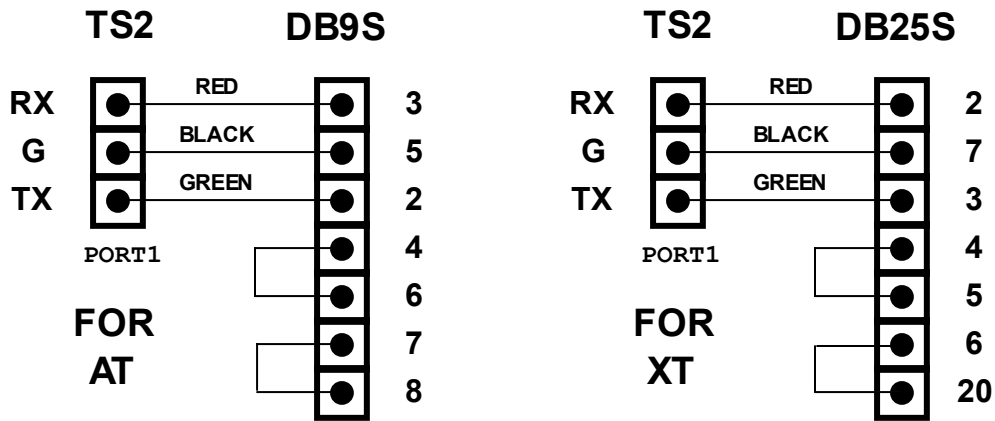


Figure 3. Serial Cable Wiring Diagrams

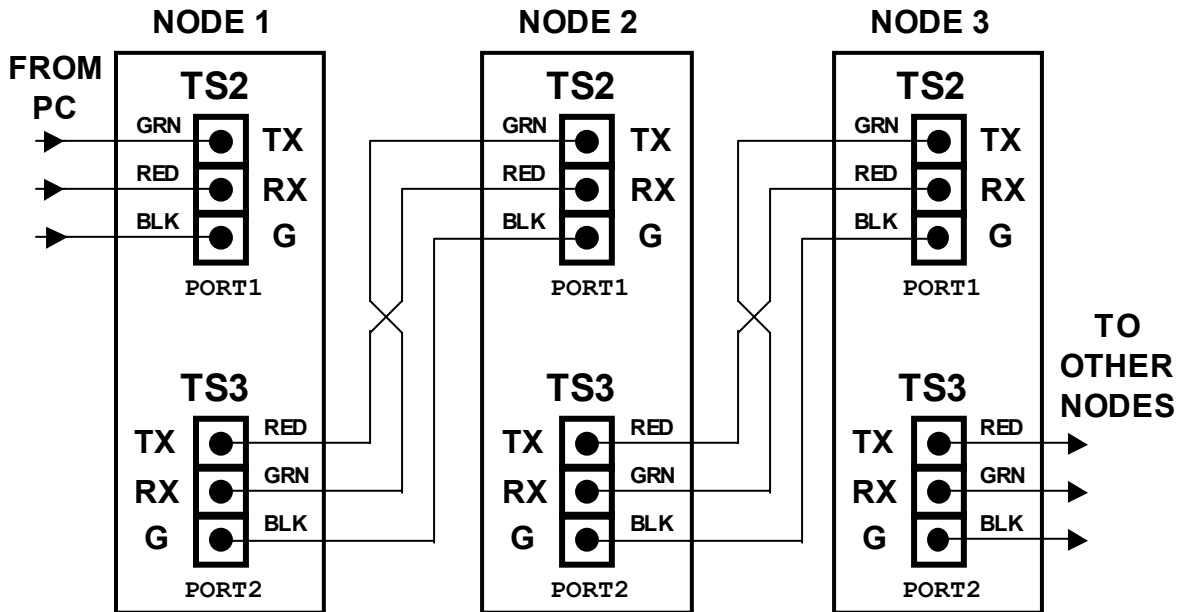


Figure 4. Multiple SSC1B Interface Circuit Boards Connected as a Network

Serial Network Interconnect Wiring

Figure 4 illustrates how multiple SSC1 system circuit boards (nodes) can be connected as a network. A 3 conductor cable simply connects from the PORT2 RS-232 terminal strip TS3 to the PORT1 terminal strip TS2 of the next SSC1 system node. The receive and transmit wiring is crossed between each node on the network as illustrated in figure 4. The RX terminal of the PORT1 terminal strip is connected to the TX terminal of the PORT2 terminal strip, and the TX terminal of the PORT1 terminal strip is connected to the RX terminal of the PORT2 terminal strip. There are no connections to the PORT2 terminal strip TS3 if it is the only node, or the last node on the network. Jumper J2, located near the center of the interface circuit board is used to configure the node for either the last or only node in the network. The jumper is placed in the upper or “L” position if it is the last or only node on the network. It is placed in the lower or “N” position if it is NOT the last or only node on the network. This jumper was pre-installed at the factory in the lower or “L” position. See the parts layout diagram near the end of this manual for the location of J2.

Network Addressing

Each node on the network has 4 hardware jumpers that are used to configure the node address. The network can have up to 16 nodes. The software that commands the network will address the nodes as 1 through 16. A hardware address between 0 and 15, corresponding to software addresses 1 through 16, is programmed with jumpers on the interface circuit board. Table 2 below indicates the proper installation of the hardware jumpers for the configuration of the node addresses. A black rectangle in table 2 indicates that a jumper is installed. Note that address 1 (software address) in table 2 has no jumpers installed and address 16 (software address) has all 4 jumpers installed at J3 positions 1, 2, 4 and 8. The SSC1B interface board is tested as node address 1 at the factory (no jumpers installed). Four header jumpers are included with the interface board for configuring the node address. A single SSC1B interface circuit board when connected properly to a serial port will function correctly as long as the commands from the host PC address it correctly. Care must be taken when configuring the hardware jumpers to insure that there are no duplicate addresses in the network. **Duplicate addresses will present problems** with the communications because some commands send back responses to the host PC after the command is received. The data would be received from 2 different nodes simultaneously and would be corrupt. The node addresses for the network do not have to be in sequence, but it would be more convenient if they were. Communications problems on the network can also result from addressing a node that does not exist on the network. Duplicate or skipped addresses on the network will not cause any physical damage to the SSC1B interface circuit boards.

		NODE ADDRESSES															
J3	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	
1		■		■		■		■		■		■		■		■	
2			■	■			■	■			■	■			■	■	
4					■	■	■	■					■	■	■	■	
8									■	■	■	■	■	■	■	■	

Table 2. Node Address Hardware Configurations

Each SSC1B interface circuit board controls the motion of 1 stepper motor and the on/off states of 4 general purpose digital outputs. It also reads the states of 4 general purpose digital inputs. Commands that address stepper motors for the purpose of motion control, and commands like “Reset Node” that affect the individual nodes, use the address range of 1 through 16. Commands that address the general purpose digital I/O use the address range of 1 through 64. For example: If you wanted to change the state of digital output 13 then the “Change Output” command would address output 13 directly. Although digital output 13 on the network is actually digital output 1 on node 4 in the network, you would address the digital output with address 13 directly. This same addressing scheme applies to both digital inputs and outputs. For the purpose of motion control synchronization of 2 or more stepper motors on a network, a node has a global flag that can be set or cleared with the “Change Global” command. When a node’s global flag is set, it will execute motion control commands with no regard to the address included in the command string. This allows you to start and stop more than 1 stepper motor at the exact same time. All stepper motor motion control commands are affected by the global flag, so you can load the same step counts to a selected number of stepper motors, or change directions, speeds etc. to a multiple number of steppers using a single command.

The SSC1 System Command Summary (SSC1B Interface)

There are a total of 18 commands in the SSC1 system. A command string consists of 2, 3 or 4 bytes that are sent to the SSC1 system for interpretation. The first byte in the command string is the command number. The valid command number range is 1 through 19. Command number 15 is not used and will be ignored by the SSC1 system. Commands 20 through 24 are 2 byte command strings that are reserved for future use with other yet undeveloped components of the SSC1 system. The SSC1B interface circuit board will interpret them as 2 byte command strings and no action will be taken. All other commands are invalid and will be ignored by the system. Depending on the type of command, the second byte in the command string may either contain the node address (1 thru 16), the stepper motor number (1 thru 16), or the digital I/O number (1 thru 64) in the network. Commands sent to the network that set step counts, speed and other information send a 3rd and possibly a 4th byte to a node for interpretation.

To eliminate the need for hardware modifications to digital input logic external to the SSC1B interface circuit board, the logic state of a digital input can be inverted using the “Change Logic” command. This will allow a normally open switch contact connected to a digital input to represent either a logic “0” or “1” state. The 4 general purpose digital inputs can also be configured as mechanical sense inputs with the use of the “Change Sense” command to stop a stepper motor. The stepper motor will stop immediately when it encounters a digital input that has its sense turned on.

Command 1 ... Reset Node/Acknowledge Node

Reset Node

2 byte string, 1 = command #, 2 = (node # in bits 0-4) (bits 5-7 cleared)

A node is reset with this command. This command is 2 bytes in length. The first byte in this command string contains the command number. The second byte of the command string contains the node address. The node that is reset will respond immediately by sending its address to the host PC.

Acknowledge Node

2 byte string, 1 = command #, 2 = (node # in bits 0-4) (acknowledge flag bit 7 set)

A node is tested with this command. This command is 2 bytes in length. The first byte in this command string contains the command number. The second byte of the command string contains the node address in bits 0 thru 4 and bit 7 is set indicating that it is the acknowledge command and not a reset. This command

does not reset the node, but the node does send its address to the host PC in response. Bit 7 is also set in the return address byte in order to distinguish it as an acknowledge response and not a reset response. You can use this command to verify communications with a node.

Command 2 ... Initialize Stepper

2 byte string, 1 = command #, 2 =(stepper # in bits 0-4) (initialize flags in bits 5-7)

The stepper motor mode and limit logic is initialized with this command. Initialization must be performed after every interface power-up or reset. The motor drive outputs will not be enabled until initialization has been performed by this command. The SSC1B interface can be re-initialized whenever the stepper motor is not in motion. Initialization commands that are received by the interface while the stepper motor is in motion will be ignored. This command is 2 bytes in length. The first byte in this command string contains the command number. The second byte contains the stepper motor number in the lower 5 bits (bits 0-4) and the 3 initialization flags in the upper 3 bits (bits 5-7). The following is a functional description of each initialization flag bit:

Initialization Flag Bit Functions

BIT 5: MS1 FLAG. (Mode Select 1) These 2 flags select the microstep resolution mode. See **BIT 6: MS2 FLAG.** (Mode Select 2) table 2 below to determine the proper mode selection.

MS1	MS2	MICROSTEP RESOLUTION
L	L	Full Step (2 Phase)
H	L	Half Step
L	H	Quarter Step
H	H	Eighth Step

Table 2. Microstep Resolution Mode Selection

BIT 7: LIMIT LOGIC FLAG. This flag is for the selection of the desired logic state of the mechanical limit sense inputs. There are 2 digital inputs on each node that are dedicated to detecting the clockwise and counter-clockwise mechanical limits for the stepper motor travel. The stepper motor will stop the instant that it encounters a mechanical limit. If you reverse the direction of travel with command 7 (Set Direction) after the limit is encountered, the motor will be allowed to move away from the mechanical limit. These limit inputs are located at terminal strip TS1 near the lower left edge of the SSC1B interface circuit board. The terminals are labeled “CW” for clockwise and “CCW” for counter-clockwise. Their common ground connection is labeled “G”. The flag in bit 7 determines the desired logic true states of the mechanical limit sense inputs. If bit 7 is cleared then the mechanical limit inputs will report an at limit condition when a limit input state is high. If bit 7 is set then the mechanical limit inputs will report an at limit condition when a limit input goes low. The mechanical limit digital inputs have pull-up resistors so they will always be at a logic high state when there is nothing connected to them, or when an open switch contact is connected to them. If the mechanical limit inputs are not going to be used then bit 7 should be set when initialized so that the limit inputs will be effectively disabled. If bit 7 is cleared when initialized and the mechanical inputs are not going to be used then a wire jumper will have to be installed on the limit inputs to ground to disable them.

Command 3 ... Change Global (Global)

2 byte string, 1 = command #, 2 = (bits 0-4 = stepper #) (bit 6 = all nodes flag) (bit 7 = global flag)

The global flag is used in conjunction with the stepper motor motion control commands. It was implemented into the system so that a single command can be simultaneously executed by a selected group of stepper motors. You may want to start or stop a group of stepper motors on the network at the exact same time with one command. Or, you may want to load the same step count for a selected group of stepper motors. The global flag is used for that purpose. Each stepper motor controller (node) has a global mode flag that can be set or cleared with this command. This command is 2 bytes in length. The first byte in this command string contains the command number. The second byte contains the stepper motor number in the lower 5 bits (bits 0 thru 4) and the desired global flag state in the most significant bit (bit 7). When this command's global flag bit is set, the node or stepper motor that is addressed by the command will set its global flag and thereafter will execute any of the designated global commands regardless of the stepper address that is contained in the command string. When this command's global flag bit is cleared, the stepper motor that is addressed by the command will clear its global flag and thereafter will only execute commands that are addressed to it directly. Every motion control command that is sent includes a stepper motor number as the second byte in the command string. As always, the stepper motor that is addressed will execute the command. Other stepper motors that have their global flags set will also execute the command simultaneously. Commands that can be executed globally are indicated as global after the command name in this command summary section of the technical manual. The "Change Global" command is by itself global in nature because all nodes can have their global flags set or cleared simultaneously if bit 6 is set in the second byte of the command string. If bit 6 is set then all nodes will have their global flags set or cleared depending on the state of bit 7. This is useful for setting or resetting the states of the global flags for all the nodes on the network simultaneously. The global flag bit state is cleared after a power-up or reset.

Command 4 ... Set Speed (Global)

3 byte string, 1 = command #, 2 = stepper #, 3 = Speed

This command sets the stepper motor(s) speed. This command is 3 bytes in length. The first byte of this command string contains the command number. The second byte contains the stepper motor number and the third byte contains the stepper motor speed setting. The stepper speed setting has a range of 1 to 255 and represents the number of steps-per-second multiplied by 10. A speed setting of 1 will set the step rate to 10 steps-per-second and a speed setting of 255 will set the step rate to 2550 steps-per-second. A setting of 0 is invalid and the node will set a default setting of 1 if a 0 is received. A new speed setting can be set for the next move while the stepper motor is currently in motion. The new stepper speed setting will not be used until a new run or start command is sent, or an auto run or auto start command is executed. Any node that is addressed by this command, or has its global flag set, will have its stepper speed set with this command. The stepper speed is set to 10 after a power-up or reset.

Command 5 ... Set Divisor (Global)

3 byte string, 1 = command #, 2 = (stepper # in bits 0-4) (HiDivisor in bits 6-7), 3 = LoDivisor

This command sets the stepper motor(s) speed divisor. This command is 3 bytes in length. The first byte of this command string contains the command number. The second byte contains the stepper motor number and the 2 most significant bits of the 10-bit speed divisor. The 5-bit stepper number is contained in bits 0 thru 4 and the 2 most significant bits (bits 8 and 9) of the 10-bit speed divisor is contained in bits 6 and 7. The third byte contains the 8 least significant bits (bits 0 thru 7) of the 10-bit speed divisor. The speed divisor setting has a range of 1 to 1023, and it represents the division of the stepper motor's speed setting. A setting of 0 is invalid and the node will set a default setting of 1 if a 0 is received. A new speed divisor

setting can be set for the next move while the stepper motor is in motion. The new stepper speed setting will not be used until a new run or start command is sent, or an auto run or auto start command is executed by the command interpreter. Any node that is addressed by this command, or has its global flag set, will have its speed divisor set by this command. The stepper speed divisor setting is 10 after a power-up or reset.

Command 6 ... Set Step Count (Global)

4 byte string, 1 = command #, 2 = (stepper # in bits 0-4) (HiSteps in bits 6-7), 3 = MdSteps, 4 = LoSteps

This command sets the stepper motor(s) step count. This command is 4 bytes in length. The first byte of this command string contains the command number. The second byte contains the stepper motor number in the lower 5 bits (bits 0 thru 4) and the 2 most significant bits of the 18-bit step count in the 2 upper bits (bits 6-7). The third and fourth bytes contain the lower 16 bits (bits 0 thru 15) of the 18-bit step count. Bits 8 through 15 of the step count are contained in the third byte, and bits 0 through 7 of the step count are contained in the fourth byte. The step count range is from 1 to 262,143. A new step count can be set for the next move while the stepper motor is in motion. The new step count will not be used until a new run or start command is sent, or an auto run or auto start command is executed by the command interpreter. Any node that is addressed by this command, or has its global flag set, will have its step count set by this command. The step count setting is 0 after a power-up or reset.

Command 7 ... Change Direction (Global)

2 byte string, 1 = command #, 2 = (stepper # in bits 0-4) (direction flag in bit 7)

This command changes the stepper motor(s) direction. This command is 2 bytes in length. The first byte in this command string contains the command number. The second byte contains the stepper motor number in the lower 5 bits (bits 0-4) and the step direction flag in the upper bit (bit 7). If the direction flag is cleared then the direction will be clockwise. If the direction flag is set then the direction will be counter-clockwise. The direction can be changed while the stepper motor is in motion. The new direction will not be used until a new run or start command is issued, or an auto run or auto start command is executed by the command interpreter. Any node that is addressed by this command, or has its global flag set, will have its direction changed by this command. The direction flag bit is cleared for clockwise direction after a power-up or reset.

Command 8 ... Run Stepper/Auto Run Stepper (Global)

Run Stepper

2 byte string, 1 = command #, 2 = (stepper # in bits 0-4)

This command starts the stepper(s) in motion. This command does not execute any steps contained in the step count register. When a stepper motor is sent into motion with this command it will continue to run until it reaches a mechanical limit or an active sense input, or the stop command is received. This command is 2 bytes in length. The first byte in this command string contains the command number and the second byte contains the stepper motor number. Any node that is addressed by this command, or has its global flag set, will start the stepper into motion if the following conditions exist: (1) the stepper motor is not already in motion, (2) it is not at a mechanical limit, (3) it is not at an active sense input. The run flag bit state in the stepper status word will be set whenever the stepper motor is in motion. The run flag bit is cleared after a power-up or reset.

Auto Run Stepper

2 byte string, 1 = command #, 2 = (stepper # in bits 0-4) (bit 7 set)

The format and functionality for the auto run command is the same as the run command as described above with the exception that bit 7 is set in the second byte of the command string containing the stepper motor number. After this command is received the stepper motor(s) will automatically start the next move using previously loaded motor control parameters immediately after it is finished running from the last move. The PLOAD status flag indicates when an auto run or auto start command is pending. See the description of the Read Stepper Status command for the function of the PLOAD status flag. The auto run command can be used as a replacement to the run command. The run command is included with this command interpreter for the purposes of maintaining downwards compatibility with the SSC1 stepper motor control system.

Command 9 ... Start Stepper/Auto Start Stepper (Global)

Start Stepper

2 byte string, 1 = command #, 2 = (stepper # in bits 0-4)

This command starts the stepper(s) in motion executing a selected step count. This command is 2 bytes in length. The first byte in this command string contains the command number and the second byte contains the stepper motor number. If a new step count has been loaded with the Set Step Count command since the last Run Stepper or Start Stepper command execution then it will be selected for use with this command. If a new step count has not been loaded and the position complete flag is set from a previous Run Stepper or Start Stepper command execution then the previously loaded step count will be selected for use with this command. If a new step count has not been loaded and the position complete flag is not set then the number of steps remaining from the last Run Stepper or Start Stepper command execution will be selected for use with this command. If the number of steps remaining is zero then the previously loaded step count will be used. After all the steps are executed, the motor will stop automatically, the position complete flag will be set, and the run flag will be cleared. Any node that is addressed or has its global flag set will start the stepper into motion if the following conditions exist: (1) the stepper motor is not already in motion, (2) it is not at a mechanical limit, (3) it is not at an active sense input, (4) the step count does not equal 0. The run flag bit state in the stepper status word will be set whenever the stepper motor is in motion. The position complete flag bit in the stepper status word will be set only after the stepper motor has executed the steps contained in its step count register. If a stepper motor is placed into motion with this command, the position complete flag **WILL NOT BE SET** if the stepper motor is stopped by the Stop Stepper command, or it is stopped because it has reached a mechanical limit or an active sense input.

Auto Start Stepper

2 byte string, 1 = command #, 2 = (stepper # in bits 0-4) (bit 7 set)

The format and functionality for this command is the same as the start command as described above with the exception that bit 7 is set in the second byte of the command string containing the stepper motor number. After this command is received the stepper motor(s) will automatically start the next move using previously loaded motor control parameters immediately after it is finished running from the last move. The PLOAD status flag indicates when an auto run or auto start command is pending. See the Read Stepper Status command description for the explanation of this flag. The auto start command can be used as a replacement to the start command. The start command is included with this command interpreter for the purposes of maintaining downwards compatibility with the SSC1 stepper motor control system.

Command 10 ... Stop Stepper (Global)

2 byte string, 1 = command #, 2 = (stepper #)

This command stops the stepper(s). This command is 2 bytes in length. The first byte in this command string contains the command number and the second byte contains the stepper motor number. Any node that is addressed, or has its global flag set and is not already stopped, will be stopped with this command. Any Auto Start or Auto Run commands that are pending execution will be canceled with this command. If a stepper motor was sent into motion with the Run Stepper command and this command is used to stop the stepper motor, the position complete flag bit in the stepper status word **WILL BE SET**. If a stepper motor was sent into motion with the Start Stepper command and this command is used to stop the stepper motor, the position complete flag bit in the stepper status word **WILL NOT BE SET**. The run flag bit state will be cleared by this command.

Command 11 ... Get Stepper Status

2 byte string, 1 = command #, 2 = (stepper # in bits 0-4) (bit 7= 0 for word 1, bit 7 = 1 for word 2)

This command reads the stepper status words of a stepper motor. You can read the stepper status words at any time, even while the stepper motor is in motion. This command is 2 bytes in length. The first byte in the command string contains the command number. The second byte contains the stepper motor number in bits 0 thru 4 and the state of bit 7 determines which status word is to be read. The 1byte status word is sent to the host PC immediately after the command is sent. If bit 7 in the second byte of the command string is cleared then status word 1 will be read. If bit 7 is set then status word 2 will be read.

The following is a functional description of the individual status word bits:

Status Word 1 Flag Bit Functions

Bit 0: POSCMP FLAG. (Position Complete) The purpose of this status flag is to determine when a stepper motor has finished moving the designated number of steps located in its step count register. The flag is set when the stepper motor is stopped automatically after executing its designated number of steps. The flag is cleared when the Set Step Count command loads a new step count, or when the Run Stepper or Start Stepper commands are executed. If a stepper motor was placed into motion with the Start Stepper command, the position complete flag will remain cleared if the stepper motor is stopped because it has reached a mechanical limit or an active sense input, or it has been stopped by the Stop Stepper command. The position complete flag will be set if a stepper motor was placed into motion with the Run Stepper command and it is stopped with the Stop Stepper command. The position complete flag bit is set after a power-up or reset.

Bit 1: RUN FLAG. The purpose of this status flag is to determine whether a stepper motor is in motion or is stopped. It is set when it is in motion and it is cleared when it is stopped. The Run flag bit is cleared after a power-up or reset.

Bit 2: LCCW FLAG. (Limit, Counter-clockwise) The purpose of this status flag is to indicate that a stepper motor has reached the counter-clockwise mechanical limit of travel. It is set when the stepper motor is at the counter-clockwise limit and cleared when it is not.

Bit 3: LCW FLAG. (Limit, Clockwise) The purpose of this status flag is to indicate that a stepper motor has reached the clockwise mechanical limit of travel. It is set when the stepper motor is at the clockwise limit and cleared when it is not.

Bit 4: SENSE FLAG. The purpose of this flag is to indicate that a stepper motor has been automatically stopped at a digital input that has input sense enabled. It is set whenever a stepper motor has been stopped at an active sense input. The stepper motor cannot move in either direction if it is stopped by an active sense input until the sense flag is cleared. The flag is cleared whenever the Change Sense command turns off input sense for the digital input that has stopped the motor, or the digital input that has stopped the motor has its logic state reversed using the Change Logic command. There is only 1 sense flag for the 4 digital inputs on a node. Your application program must keep track of which inputs have their sense activated. If sense is turned on for more than one input, you can read the states of all 4 digital inputs on the node to determine which input the motor has stopped at.

Bit 5: OEBIT FLAG. (Overrun Error Bit) The purpose of this flag is to indicate that a serial port overrun error has occurred. The flag will be set if an overrun error has occurred. It will be cleared when the status word is fetched with the Get Status command.

Bit 6: FEBIT FLAG. (Framing Error Bit) The purpose of this flag is to indicate that a serial port framing error has occurred. The flag will be set if a framing error has occurred. It will be cleared when the status word is fetched with the Get Status command.

Bit 7: GLB FLAG. (Global Flag) The purpose of this flag is to indicate the state of a nodes global flag. The flag is set or cleared with the Change Global command.

Status Word 2 Flag Bit Functions

Bit 0: PLOAD FLAG. (Parameters Load) The purpose of this flag is to indicate if an auto run or auto start command is pending. The PLOAD flag is cleared when an auto run or auto start command is received. The PLOAD flag is set when a run or start command is sent, or when an auto run or auto start command is executed by the command interpreter. Stepper motor control parameters for the next move can be safely loaded while PLOAD is set. The loading of stepper motor parameters while PLOAD is cleared is not safe because a pending auto run or auto start command can occur at any time. When a run, auto run, start, or auto start command is executed the preliminary parameters will be transferred to the active parameters area just before the move begins. It is recommended that the application software always load stepper motor control parameters while PLOAD is set.

Bit 1: DIR FLAG. (Direction) The purpose of this flag is to indicate the current direction setting. If the stepper status word is read while the motor is running then the active direction flag state will be sent. If the motor is not running then the preliminary direction flag state will be sent. DIR is cleared for clockwise direction and set for counter-clockwise direction.

Bit 2: INIT FLAG. (Initialization) The purpose of this flag is to determine whether the stepper motor has been initialized. The INIT flag is cleared if command 2 (Initialize Stepper) has not been sent following a power-up or reset. The INIT flag is set if the stepper has been initialized.

Bit 3: CONT FLAG. (Continuous Run) The purpose of this flag is to indicate that a stepper motor is in the continuous run mode. The CONT flag is cleared if the motor is not in run mode. The CONT flag is set if it is in run mode.

Bit 4: AEN FLAG. (Auto Drive Enable) The purpose of this flag is to indicate the state of the auto drive enable mode. If AEN is cleared then auto drive enable is off. If AEN is set then auto drive enable is on.

Bit 5: PWRDWN FLAG. (Power Down) The purpose of this flag is to indicate the state of the power down mode. The power down mode is off when the PWRDWN flag is cleared. Power down mode is on when PWRDWN is set.

Bit 6: ACCEL FLAG. (Acceleration) The purpose of this flag is to indicate whether ramp acceleration is turned on or off. If ACCEL is cleared then ramp acceleration is turned off. If ACCEL is set then ramp acceleration is turned on. If the stepper status word is read while the motor is running then the active ACCEL flag state will be sent. If the motor is not running then the preliminary ACCEL flag state will be sent.

Bit 7: DECEL. (Deceleration) The purpose of this flag is to indicate whether ramp deceleration is turned on or off. If DECEL is cleared then ramp deceleration is turned off. If DECEL is set then ramp deceleration is turned on. If the stepper status word is read while the motor is running then the active DECEL flag state will be sent. If the motor is not running then the preliminary DECEL flag state will be sent.

Command 12 ... Get Steps Remaining/Get Steps Taken

Get Steps Remaining

2 byte string, 1 = command #, 2 = (stepper # in bits 0-4) (bit 5 cleared)

This command reads the current steps remaining for a stepper motor. The remaining step count can be read at any time, even while the stepper motor is in motion. This command is 2 bytes in length. The first byte in the command string contains the command number. The second byte contains the stepper motor number in bits 0 thru 4 and **bit 5 is cleared**. The 3 bytes containing the 18 bit step count are sent to the host PC immediately after the command is sent. The high byte of the step count containing the 2 most significant bits (bits 16-17) is sent first followed by the middle byte (bits 8-15) and the lower byte (bits 0-7). The active steps remaining will be sent if the stepper motor is running when this command is sent. The preliminary steps remaining will be sent if the stepper motor is not running and a new step count has been loaded when this command is sent. The active steps remaining will be sent if the motor is stopped and a new step count has not been loaded. The example program listing after this command summary illustrates how the 3 step count bytes can be combined into one 18-bit value.

Get Steps Taken

2 byte string, 1 = command #, 2 = (stepper # in bits 0-4) (bit 5 set)

The format for this command is identical to the Get Steps Remaining command as described above with the exception that **bit 5 is set** in the second byte in the command string containing the stepper motor number. This command reads the current number of steps taken for a stepper motor. The number of steps taken can be read at any time, even while the stepper motor is in motion. The active steps taken will be sent if the stepper motor is running when this command is sent. The steps taken will read as 0 if the motor is not running and a new step count has been loaded.

Command 13 ... Set Ramp Parameters (Global)

Set Ramp Rate (Global)

3 byte string, 1 = command #, 2 = (stepper # in bits 0-4) (bits 5, 6, 7 clear), 3 = Ramp rate variable

This command sets the ramp rate for a stepper motor. This command is 3 bytes in length. The first byte in the command string contains the command number. The second byte contains the stepper motor number in bits 0 through 4 and **bits 5, 6 and 7 cleared**. The third byte contains the ramp rate setting. The ramp rate setting has a range of 1 to 255. If the node receives a setting of 0 it will store a ramp rate setting of 1. The ramp rate setting limits the stepper speed of the first 255 steps taken during acceleration, or the last 255 steps remaining during deceleration. The longest possible ramp, which is 255 steps, is obtainable by setting the stepper speed value to 255 and the ramp rate value to 1. The number of steps in the ramp is determined by dividing the stepper speed value by the ramp rate value. The stepper speed divisor has no influence on ramping and is only used to lower the step rate by a selected division of the stepper speed. Ramping works best when the stepper speed is at a high setting and the stepper speed divisor is used to set the desired motor speed. The ramp offset and ramp stretch commands explained in detail below can be used for additional ramp modifications. The ramp rate value for the next move can be set when the stepper motor is in motion. The new ramp rate value will not be used until a new run or start command is sent, or an auto run or auto start command is executed by the command interpreter. Any stepper motor that is addressed by this command, or has its global flag set, will have its ramp rate value set. The ramp rate value is set to 255 after a power-up or reset.

Command 13 ... Set Ramp Flags (Global)

3 byte string, 1 = command #, 2 = (stepper # in bits 0-4, bit 5 set, bit 6, 7 clear), 3 = Flags in bits 0, 1

This command sets the ramp accelerate and decelerate flag states for a stepper motor. The format for this command is identical to the Set Ramp Rate command as described above with the exception that **bit 5 is set and bits 6 and 7 are cleared** in the second byte of the command string, and the third byte contains the accelerate flag in bit 0 and the decelerate flag in bit 1. If both flag bits are cleared ramping is turned off and the stepper speed will be constant at the stepper speed setting. If both flag bits are set ramping will be turned on for both acceleration and deceleration. If the accelerate flag bit is the only bit set then only acceleration ramping will occur during stepper motion. If the deceleration flag bit is the only bit set then only deceleration ramping will occur. Any stepper motor that is addressed by this command, or has its global flag set, will set the ramp flag bits. The accelerate and decelerate ramp flags are both set after a power-up or reset.

Command 13 ... Set Ramp Offset (Global)

3 byte string, 1 = command #, 2 = (stepper # in bits 0-4) (bit 6 set, bits 5,7 clear), 3 = Ramp offset variable

This command sets the ramp offset variable for a stepper motor. The format for this command is the same as the Set Ramp Rate and Set Ramp Flags commands described above with the exception that **bit 6 is set and bits 5 and 7 are cleared** in the second byte of the command string, and the third byte contains the ramp offset value. The ramp offset value is added to the ramp rate value when a new speed is calculated before each step that is taken. The ramp offset sets the minimum speed at the beginning of acceleration and at the end of deceleration. The maximum number of steps in the ramp decreases proportionately by the ramp offset value. The ramp offset value has a range of 0 to 255. Any stepper motor that is addressed by this command, or has its global flag set, will have its ramp offset value set. The ramp offset value is set to 0 after a power-up or reset.

Command 13 ... Set Ramp Stretch flags (Global)

3 byte string, 1 = command #, 2 = (stepper # in bits 0-4) (bits 5,6 clear, bit 7 set), 3 = flags in bits 0, 1

This command sets the ramp stretch flags for a stepper motor. The format for this command is the same as the Set Ramp Parameters commands as described above with the exception that **bits 5 and 6 are cleared and bit 7 is set** in the second byte of the command string and the third byte contains the ramp times 2 flag in bit 0 and the ramp times 4 flag in bit 1. If both flag bits are cleared ramping will be normal at a maximum of 255 steps. The ramp will be multiplied 2 times for a maximum of 512 steps if the ramp times 2 flag (bit 0) is set. The ramp will be multiplied 4 times for a maximum of 1024 steps if the ramp times 4 flag (bit 1) is set. The ramp will be multiplied 4 times if both flag bits are set. Any stepper motor that is addressed by this command, or has its global flag set, will have its ramp stretch flag bits set. The ramp stretch flag bits are cleared after a power-up or reset.

Command 14 ... Change Auto/Change Power (Global)

Change Auto

2 bytes, byte 1 = command, byte 2 = (bits 0-4 = stepper #) (bits 5,6 clear, bit 7 = flag)

This command allows you to enable or disable the auto drive feature. Auto drive can be changed at any time, even while the stepper motor is in motion. The command is 2 bytes in length. The first byte in the command string contains the command number. The second byte contains the stepper motor number in the lower 5 bits (bits 0-4), **bits 5 and 6 are cleared**, and the auto drive enable flag state is in the upper bit (bit 7). The term "AUTO" is used for this command because the motor can still be sent into motion if the drive outputs are disabled with this command. When auto drive is enabled all motor drive outputs will automatically be turned off 1 step time after the stepper motor stops motion, and all motor drive outputs will be turned back on again 1 step time before it starts into motion. The motor drive outputs will be enabled or disabled instantly with this command if the motor is stopped. This command is useful for disabling the motor drive circuits to conserve power and to prevent heat buildup when the motor will be idle for long periods of time. However, some power may be necessary for braking or torque lock while the motor is stopped. The Change Power command described below can lower the drive power while the motor is stopped. Auto drive will be enabled if bit 7 is set and disabled if bit 7 is cleared. Any stepper motor that is addressed by this command, or has its global flag set, will have its auto drive setting changed. The auto drive enable flag is cleared after a power-up or reset.

Command 15 ... Change Power

The format for this command is identical to the Change Auto command as described above with the exception that **bit 5 is set and bit 6 is cleared** in the second byte of the command string, and the power level control flag state is contained in the upper bit (bit 7). If the power level control flag state in bit 7 is cleared then stepper motor drive power output level will be at its normal level at all times. If the power level control flag bit state is set then motor power will be decreased to approximately one third of its normal power level 2 seconds after the motor has stopped. It will resume its normal power level as soon as it is commanded to move again. Any stepper motor that is addressed by this command, or has its global flag set, will have its power level flag setting changed. The power level flag is cleared after a power-up or reset.

Command 16 ... Get Input

2 bytes, byte 1 = command, byte 2 = input #

This command reads the state of a digital input on the network. The command is 2 bytes in length. The first byte of the command string contains the command number and the second byte contains the digital input number. There are 4 digital inputs per node. Therefore, there can be up to 64 digital inputs in a network. The valid digital input address range for this command is from 1 to 64. A digital input is addressed directly with this command. Although digital input 13 in a network is actually digital input 1 of node 4 in a network, you would address it using input number 13 directly. The input state byte is sent to the host PC immediately after this command is sent. The input state byte contains the latched and current states of the digital input. The current logic state of the digital input is contained in bit 0 and the latched logic state is contained in bit 4. The current input bit 0 state is set to whatever the actual digital input state is when the command is received. The latched input bit 4 state is set whenever a digital input goes high and remains set until the digital input state is read by the host PC with this command. Command 18 can reverse (invert) the logic of a digital input. See the description of command 18 for details. The subroutine GetInput in the example program listing that follows this command summary shows you how to extract the current and latched bit states of the digital input byte received and create separate variables for them.

Command 17 ... Change Output

2 bytes, byte 1 = command, byte 2 = (bits 0-4 = input #) (bit 7 = flag)

This command changes the logic state of a digital output on the network. The command is 2 bytes in length. The first byte in the command string contains the command number and the second byte contains the digital output number in the lower 5 bits (bits 0 thru 4) and the desired output logic state in the upper bit (bit 7). There are 4 digital outputs per node. Therefore, there can be up to 64 digital outputs in a network. The valid digital output address range for this command is from 1 to 64. A digital output is addressed directly with this command. Although digital output 17 in a network is actually digital output 1 on node 5 in a network, you would address it using output number 17 directly.

Command 18 ... Change Logic

2 bytes, byte 1 = command, byte 2 = (bits 0-4 = input #) (bit 7 = flag)

This command allows you to change the logic state of a digital input. The command is 2 bytes in length. The first byte of the command string contains the command number. The second byte contains the digital input number in the lower 5 bits (bits 0-4) and the desired input logic state in the upper bit (bit 7). The valid addressing range is from 1 to 64, and the digital input is addressed directly. If the state of bit 7 is cleared then the digital input will be read as a normal logic 1 state when the digital input is at a logic 1 state. If the state of bit 7 is set then the digital input logic will be reversed reading a logic 0 state when the digital input is actually at a logic 1 state. The digital inputs have pull-up resistors. Therefore, they will be read as a logic high state when there is nothing connected to them. You can make them read a logic 0 state with this command. This command is useful when a normally open switch contact or other devices that normally indicate a logic high state is connected to a digital input. The host PC cannot read the states of the input logic flags. Your application program will have to keep track of their current states. The input logic state flags will be cleared after a power-up or reset.

Command 19 ... Change Sense

2 bytes, byte 1 = command, byte 2= (bits 0-4 = input # (bit 7 = flag)

This command allows you to enable or disable input sense for a digital input. The command is 2 bytes in length. The first byte of the command string contains the command number. The second byte contains the digital input number in the lower 5 bits (bits 0-4) and the desired input sense state in the upper bit (bit 7). A stepper motor will stop motion immediately when a sense enabled digital input goes to a logic high state. The stepper motor will not be allowed to move again until input sense is disabled for the digital input that caused it to stop. The valid addressing range is from 1 to 64, and the digital input is addressed directly. If the state of bit 7 is cleared then input sense will be disabled. If the state of bit 7 is set then input sense will be enabled. The host PC cannot read the states of the input sense flags. Your application program will have to keep track of their current states. The digital input sense flags are cleared after a power-up or reset.

Program Example

The example QuickBasic program of Listing 1 is a stepper motor exerciser program that utilizes all of the SSC1B system commands. It is intended to illustrate how easy it is to command and control a stepper motor and the digital I/O in the system. This program (SSC1BEXR.EXE) is also included on the programming examples disk that comes with the system. The QuickBasic file (SSC1BEXR.BAS) is also included on the disk in text format. It is important to note that the QuickBasic programming examples will not run properly with the Windows XP operating system. Attempting to run the QuickBasic programming examples with Windows XP will produce Device I/O errors. The QuickBasic programming examples should run properly with all older versions of the Windows operating system. The Visual Basic programming examples that are included on the programming examples disk will run with all Windows operating systems.

Listing 1. A Simple Stepper Motor and I/O Exerciser Program for the SSC1B Interface.

```
DECLARE SUB GetStatus (StepperNumber!, Byte2Flag!, Bit0, Bit1!, Bit2!, Bit3!, Bit4!, Bit5!, Bit6!, Bit7!)
DECLARE SUB ChangeAuto (StepperNumber!, AutoEnable, AutoPower!)
DECLARE SUB GetInput (InputNumber!, NowValue!, LatchValue!)
DECLARE SUB GetStepsRemaining (StepperNumber!, StepsRemaining&)
DECLARE SUB GetStepsTaken (StepperNumber!, StepsTaken&)
DECLARE SUB StartStepper (StepperNumber!, AutoStart!)
DECLARE SUB StopStepper (StepperNumber!)
DECLARE SUB SetStepCount (StepperNumber!, StepCount&)
DECLARE SUB ChangeGlobal (Node!, GlobalState!)
DECLARE SUB MoveCursor ()
DECLARE SUB ReadInput (InputNumber!, NowValue!, LatchValue!)
DECLARE SUB ChangeOutput (OutputNumber!, OutputState!)
DECLARE SUB SetDirection (StepperNumber!, DIR!)
DECLARE SUB RunStepper (StepperNumber!, AutoRun!)
DECLARE SUB GetStatus (StepperNumber!, Byte2Flag!, Bit0!, Bit1!, Bit2!, Bit3!, Bit4!, Bit5!, Bit6!, Bit7!)
DECLARE SUB ChangeSense (InputNumber!, SenseState!)
DECLARE SUB SetRamp (StepperNumber!, RampRate!)
DECLARE SUB SetRampStretch (StepperNumber!, SFlags!)
DECLARE SUB SetOffset (StepperNumber!, RampOffset!)
DECLARE SUB SetAccelDecel (StepperNumber!, RFlags!)
DECLARE SUB ChangeLogic (InputNumber!, InputLogicState!)
DECLARE SUB SetSpeed (StepperNumber!, Speed!)
DECLARE SUB SetDivisor (StepperNumber!, Divisor%)
DECLARE SUB Initialize (StepperNumber!, StepMode!, LimitLogic!)
DECLARE SUB ResetNode (Node!, ResetResponse!)
```

```

'***** STEPPER EXERCISER PROGRAM *****
'***** FOR THE SSC1B INTERFACE *****
'***** 08/04/03 *****

```

```

' Eggert Electronics Eng.
' Daniel N. Eggert
' 125 Pasa Por Aquí Ln.
' Alamogordo, NM 88310
' PH (505) 437-0698

```

```

CONST FullStep = 0, HalfStep = 32
CONST QuarterStep = 64, EighthStep = 96
CONST NormalLimitLogic = 0, InvertedLimitLogic = 1
CONST NormalInputLogic = 0, InvertedInputLogic = 1
CONST TurnSenseOff = 0, TurnSenseOn = 1
CONST GlobalOff = 0, GlobalOn = 1
CONST TurnOffOutput = 0, TurnOnOutput = 1
CONST AutoEnableOff = 0, AutoEnableOn = 1
CONST AutoPowerOff = 0, AutoPowerOn = 1

```

```

CLS : OPEN "COM1:9600,N,8,1" FOR RANDOM AS #1

```

```

DO UNTIL EOF(1) 'Make sure that the input buffer is empty!
  I = ASC(INPUT$(1, #1))
LOOP

```

```

Node = 1: StepperNumber = 1 'Default node address and stepper number
InitFlag = 0 'Initialization flag. 0 = not initialized

```

```

PRINT
PRINT "          SSC1 STEPPER EXERCISER COMMAND MENU"
PRINT
PRINT "  1 - Reset Node          11 - Read Stepper Status"
PRINT "  2 - Initialize Stepper  12 - Read Step Count"
PRINT "  3 - Change Global Flag  13 - Set Ramp parameters"
PRINT "  4 - Set Stepper Speed   14 - Change Auto Drive"
PRINT "  5 - Set Stepper Speed Divisor  15 - Not Used"
PRINT "  6 - Set Stepper Step Count  16 - Read Digital Input"
PRINT "  7 - Set Stepper Direction  17 - Change Digital Output"
PRINT "  8 - Run Stepper          18 - Change Input Logic"
PRINT "  9 - Start Stepper        19 - Change Input Sense"
PRINT " 10 - Stop Stepper         20 - Change Node/Stepper"

```

```

LOCATE 19, 23: PRINT "NODE/STEPPER Selected =" + STR$(Node) + " ";

```

```

DO 'The main program loop starts here.

```

```

CALL MoveCursor
INPUT "Enter command number "; Command

```

```

SELECT CASE Command

```

```

CASE 1 'Reset Node
  LOCATE 16, 5: PRINT STRING$(75, " ");
  CALL ResetNode(Node, ResetResponse)
  LOCATE 17, 5
  PRINT STRING$(70, " ");
  LOCATE 17, 20

```

```

IF ResetResponse = Node THEN
  PRINT "Node" + STR$(ResetResponse) + " Reset Response Received"
  InitFlag = 0
ELSE
  PRINT "Unknown Reset Response"
END IF

CASE 2 'Initialize Stepper
CALL MoveCursor
INPUT "Select mode (Full, Half, Quarter, Eighth) 1/2/4/8 "; A$
IF A$ = "1" THEN StepMode = FullStep
IF A$ = "2" THEN StepMode = HalfStep
IF A$ = "4" THEN StepMode = QuarterStep
IF A$ = "8" THEN StepMode = EighthStep
CALL MoveCursor
INPUT "Invert mechanical limits Y/n "; A$
IF A$ = "n" OR A$ = "N" THEN
  LimitLogic = NormalLimitLogic
ELSE
  LimitLogic = InvertedLimitLogic
END IF
CALL Initialize(StepperNumber, StepMode, LimitLogic)
LOCATE 17, 5: PRINT STRING$(75, " ")
LOCATE 17, 25
PRINT "Stepper initialized"
InitFlag = 1

CASE 3 'Change Global Flag
CALL MoveCursor
INPUT "Set or clear global flag S/C "; A$
IF A$ = "c" OR A$ = "C" THEN
  GlobalState = GlobalOff
ELSE
  GlobalState = GlobalOn
END IF
CALL ChangeGlobal(Node, GlobalState)

CASE 4 'Set Stepper Speed
CALL MoveCursor
DO
  CALL MoveCursor
  INPUT "Enter Stepper Speed (1-255) "; StepperSpeed
  IF StepperSpeed < 0 OR StepperSpeed > 255 THEN BEEP
LOOP WHILE StepperSpeed < 0 OR StepperSpeed > 255
CALL SetSpeed(StepperNumber, StepperSpeed)

CASE 5 'Set Stepper Speed Divisor
DO
  CALL MoveCursor
  INPUT "Enter stepper speed divisor (1-1023) "; Divisor%
  IF Divisor% < 0 OR Divisor% > 1023 THEN BEEP
LOOP WHILE Divisor% < 0 OR Divisor% > 1023
CALL SetDivisor(StepperNumber, Divisor%)

CASE 6 'Set Step Count
DO
  CALL MoveCursor
  INPUT "Enter the stepper step count (1-262143) "; StepCount&
  IF StepCount& < 1 OR StepCount& > 262143 THEN BEEP
LOOP WHILE StepCount& < 1 OR StepCount& > 262143
CALL SetStepCount(StepperNumber, StepCount&)

```

```

CASE 7 'Set Stepper Direction
DO
  CALL MoveCursor
  INPUT "Enter stepper direction 0=CW, 1=CCW "; DIR
  IF DIR < 0 OR DIR > 1 THEN BEEP
  LOOP WHILE DIR < 0 OR DIR > 1
  CALL SetDirection(StepperNumber, DIR)
  IF DIR = 0 THEN Direction$ = "CW " ELSE Direction$ = "CCW"

CASE 8 'Run Stepper
CALL MoveCursor
AutoRun = 0
INPUT "Auto or Normal Run (A,N) "; A$
IF A$ = "A" OR A$ = "a" THEN AutoRun = 1
LOCATE 17, 5: PRINT STRING$(75, " ");
IF InitFlag = 1 THEN
  CALL RunStepper(StepperNumber, AutoRun)
  LOCATE 17, 25
  PRINT "Stepper running";
ELSE
  LOCATE 17, 10
  PRINT "Command not executed because stepper is not initialized"
END IF

CASE 9 'Start Stepper
CALL MoveCursor
AutoStart = 0
INPUT "Auto or Normal Start (A,N) "; A$
IF A$ = "A" OR A$ = "a" THEN AutoStart = 1
LOCATE 17, 5: PRINT STRING$(75, " ");
IF InitFlag = 1 THEN
  CALL StartStepper(StepperNumber, AutoStart)
  LOCATE 17, 25
  PRINT "Stepper Started";
ELSE
  LOCATE 17, 10
  PRINT "Command not executed because stepper is not initialized"
END IF

CASE 10 'Stop stepper
LOCATE 17, 5: PRINT STRING$(75, " ");
IF InitFlag = 1 THEN
  CALL StopStepper(StepperNumber)
  LOCATE 17, 25
  PRINT "Stepper Stopped";
ELSE
  LOCATE 17, 10
  PRINT "Command not executed because stepper is not initialized"
END IF

CASE 11 'Get Stepper Status
CALL MoveCursor
Byte2Flag = 0
INPUT "Read Stepper Status Byte 1 or 2 (1,2) "; A$
IF A$ = "2" THEN Byte2Flag = 1

LOCATE 17, 5: PRINT STRING$(75, " ");
CALL GetStatus(StepperNumber, Byte2Flag, Bit0, Bit1, Bit2, Bit3, Bit4, Bit5, Bit6, Bit7)
LOCATE 17, 5

```

```

IF Byte2Flag = 0 THEN
  PRINT "POS =" + STR$(Bit0) + ", RUN =" + STR$(Bit1);
  PRINT ", CCL =" + STR$(Bit2) + ", CWL =" + STR$(Bit3);
  PRINT ", SEN =" + STR$(Bit4) + ", OER =" + STR$(Bit5);
  PRINT ", FER =" + STR$(Bit6) + ", GLB =" + STR$(Bit7)
ELSE
  PRINT "PLD =" + STR$(Bit0) + ", DIR =" + STR$(Bit1);
  PRINT ", INI =" + STR$(Bit2) + ", CON =" + STR$(Bit3);
  PRINT ", AEN =" + STR$(Bit4) + ", PWR =" + STR$(Bit5);
  PRINT ", ACC =" + STR$(Bit6) + ", DEC =" + STR$(Bit7)
END IF

CASE 12 'Read Step Count
CALL MoveCursor
INPUT "Steps Remaining or Taken (R, T) "; A$
LOCATE 17, 5: PRINT STRING$(75, " ");
LOCATE 17, 25
IF A$ = "R" OR A$ = "r" THEN
  CALL GetStepsRemaining(StepperNumber, StepsRemaining&)
  PRINT STR$(StepsRemaining&) + " Steps remaining"
ELSE
  CALL GetStepsTaken(StepperNumber, StepsTaken&)
  PRINT STR$(StepsTaken&) + " Steps taken"
END IF

CASE 13 'Set Ramp Parameters
LOCATE 15, 15
INPUT "Ramp Flags, Offset, Rate, Stretch (F, O, R, S) "; A$
IF A$ = "R" OR A$ = "r" THEN
  DO
    CALL MoveCursor
    INPUT "Enter Ramp Rate (0-255) "; RampRate
    IF RampRate < 0 OR RampRate > 255 THEN BEEP
    LOOP WHILE RampRate < 0 OR RampRate > 255
    CALL SetRamp(StepperNumber, RampRate)
  END IF
IF A$ = "F" OR A$ = "f" THEN
  CALL MoveCursor
  INPUT "Ramp on Accel, Decel, Both or None (A,D,B,N) "; A$
  RFlags = 0 ' Default for no ramps.
  IF A$ = "A" OR A$ = "a" THEN RFlags = 1
  IF A$ = "D" OR A$ = "d" THEN RFlags = 2
  IF A$ = "B" OR A$ = "b" THEN RFlags = 3
  CALL SetAccelDecel(StepperNumber, RFlags)
END IF
IF A$ = "O" OR A$ = "o" THEN
  DO
    CALL MoveCursor
    INPUT "Enter Ramp Offset (0-255) "; RampOffset
    IF RampOffset < 0 OR RampOffset > 255 THEN BEEP
    LOOP WHILE RampOffset < 0 OR RampOffset > 255
    CALL SetOffset(StepperNumber, RampOffset)
  END IF
IF A$ = "S" OR A$ = "s" THEN
  CALL MoveCursor
  INPUT "Stretch Ramp 1, 2 or 4 times (1,2,4) "; A$
  SFlags = 0
  IF A$ = "0" THEN SFlags = 0
  IF A$ = "2" THEN SFlags = 1
  IF A$ = "4" THEN SFlags = 2
  CALL SetRampStretch(StepperNumber, SFlags)
END IF

```

```

CASE 14 'Change Auto Drive
CALL MoveCursor
INPUT "Disable drive when idle Y/N "; A$
IF A$ = "y" OR A$ = "Y" THEN
  AutoEnable = AutoEnableOn
ELSE
  AutoEnable = AutoEnableOff
END IF
CALL MoveCursor
INPUT "Lower power when idle Y/N "; A$
IF A$ = "y" OR A$ = "Y" THEN
  AutoPower = AutoPowerOn
ELSE
  AutoPower = AutoPowerOff
END IF
CALL ChangeAuto(StepperNumber, AutoEnable, AutoPower)

CASE 15 'Not used

CASE 16 'Get Digital Input
DO
  CALL MoveCursor
  INPUT "Read digital input number (1-4) "; InputNumber
  IF InputNumber < 1 OR InputNumber > 4 THEN BEEP
  LOOP WHILE InputNumber < 1 OR InputNumber > 4
  InputNumber = InputNumber + ((Node - 1) * 4)
  LOCATE 17, 5: PRINT STRING$(75, " ");
  CALL GetInput(InputNumber, NowValue, LatchValue)
  LOCATE 17, 16
  PRINT "Digital input" + STR$(InputNumber);
  PRINT " Now Value =" + STR$(NowValue);
  PRINT ", Latched Value =" + STR$(LatchValue)

CASE 17 'Change Digital Output
DO
  CALL MoveCursor
  INPUT "Change digital output number (1-4) "; OutputNumber
  IF OutputNumber < 1 OR OutputNumber > 4 THEN BEEP
  LOOP WHILE OutputNumber < 1 OR OutputNumber > 4
  OutputNumber = OutputNumber + ((Node - 1) * 4)
  CALL MoveCursor
  INPUT "Set or clear digital output S/C "; A$
  IF A$ = "c" OR A$ = "C" THEN
    OutputState = TurnOffOutput
  ELSE
    OutputState = TurnOnOutput
  END IF
  CALL ChangeOutput(OutputNumber, OutputState)

CASE 18 'Change Digital Input Logic
DO
  CALL MoveCursor
  INPUT "Change logic state for digital input number (1-4) "; InputNumber
  IF InputNumber < 1 OR InputNumber > 4 THEN BEEP
  LOOP WHILE InputNumber < 1 OR InputNumber > 4
  InputNumber = InputNumber + ((Node - 1) * 4)
  CALL MoveCursor
  INPUT "Normal or Inverted state N/I "; A$

```

```

IF A$ = "n" OR A$ = "N" THEN
  InputLogicState = NormalInputLogic
ELSE
  InputLogicState = InvertedInputLogic
END IF
CALL ChangeLogic(InputNumber, InputLogicState)

CASE 19 'Change Digital Input Sense
DO
  CALL MoveCursor
  INPUT "Change sense for digital input number (1-4) "; InputNumber
  IF InputNumber < 1 OR InputNumber > 4 THEN BEEP
  LOOP WHILE InputNumber < 1 OR InputNumber > 4
  InputNumber = InputNumber + ((Node - 1) * 4)
  CALL MoveCursor
  INPUT "Enable or Disable sense E/D "; A$
  IF A$ = "d" OR A$ = "D" THEN
    SenseState = TurnSenseOff
  ELSE
    SenseState = TurnSenseOn
  END IF
  CALL ChangeSense(InputNumber, SenseState)

CASE 20 'Change Node/Stepper Number
DO
  CALL MoveCursor
  INPUT "Change the Node/Stepper number (1-16) "; Node
  IF Node < 1 OR Node > 16 THEN BEEP
  LOOP WHILE Node < 1 OR Node > 16
  StepperNumber = Node
  LOCATE 19, 23: PRINT "NODE/STEPPER Selected =" + STR$(Node) + " ";

END SELECT

LOOP 'This is the end of the main program loop!

END

SUB ChangeAuto (StepperNumber, AutoEnable, AutoPower)

  PRINT #1, CHR$(14); 'Send auto drive command (14)
  'Send the combined stepper number and auto drive flag
  PRINT #1, CHR$(StepperNumber + (AutoEnable * 128) + (AutoPower * 64));

END SUB

SUB ChangeGlobal (Node, GlobalState)

  PRINT #1, CHR$(3); 'Send change global command (3)
  'Send the combined node address and global flag state
  PRINT #1, CHR$(Node + (GlobalState * 128));

END SUB

SUB ChangeLogic (InputNumber, InputLogicState)

  PRINT #1, CHR$(18); 'Send change logic state command
  'Send the combined input number and invert state bit
  PRINT #1, CHR$(InputNumber + (InputLogicState * 128));

END SUB

```

```

SUB ChangeOutput (OutputNumber, OutputState)

PRINT #1, CHR$(17); 'Send change digital output command
'Send the combined output number and output state flag
PRINT #1, CHR$(OutputNumber + (OutputState * 128));

END SUB

SUB ChangeSense (InputNumber, SenseState)

PRINT #1, CHR$(19); 'Send change input sense command
'Send the combined input number and invert mask bit
PRINT #1, CHR$(InputNumber + (SenseState * 128));

END SUB

SUB GetInput (InputNumber, NowValue, LatchValue)

PRINT #1, CHR$(16); 'Send read digit input command
PRINT #1, CHR$(InputNumber); 'Send input number

ReturnData = ASC(INPUT$(1, #1))'Get input state

IF ReturnData >= 16 THEN
  LatchValue = 1
  ReturnData = ReturnData - 16
ELSE
  LatchValue = 0
END IF

IF ReturnData = 1 THEN
  NowValue = 1
ELSE
  NowValue = 0
END IF

END SUB

SUB GetStatus (StepperNumber, Byte2Flag, Bit0, Bit1, Bit2, Bit3, Bit4, Bit5, Bit6, Bit7)

PRINT #1, CHR$(11); 'Send get status command
PRINT #1, CHR$(StepperNumber + (Byte2Flag * 128)); 'Send stepper number

Status = ASC(INPUT$(1, #1))'Get the stepper status

'Preset all status bit variables to 0

Bit0 = 0: Bit1 = 0: Bit2 = 0: Bit3 = 0
Bit4 = 0: Bit5 = 0: Bit6 = 0: Bit7 = 0

IF Status >= 128 THEN
  Bit7 = 1: Status = Status - 128
END IF
IF Status >= 64 THEN
  Bit6 = 1: Status = Status - 64
END IF
IF Status >= 32 THEN
  Bit5 = 1: Status = Status - 32
END IF

```

```

IF Status >= 16 THEN
  Bit4 = 1: Status = Status - 16
END IF
IF Status >= 8 THEN
  Bit3 = 1: Status = Status - 8
END IF
IF Status >= 4 THEN
  Bit2 = 1: Status = Status - 4
END IF
IF Status >= 2 THEN
  Bit1 = 1: Status = Status - 2
END IF
IF Status >= 1 THEN Bit0 = 1
END SUB

```

```

SUB GetStepsRemaining (StepperNumber, StepsRemaining&)

```

```

  PRINT #1, CHR$(12); 'Send read steps command
  PRINT #1, CHR$(StepperNumber); 'Send stepper number
  HiByte = ASC(INPUT$(1, #1))'Get the High step byte
  MdByte = ASC(INPUT$(1, #1))'Get the Middle step byte
  LoByte = ASC(INPUT$(1, #1))'Get the Low step byte
  StepsRemaining& = (HiByte * 65536) + (MdByte * 256) + LoByte

```

```

END SUB

```

```

SUB GetStepsTaken (StepperNumber, StepsTaken&)

```

```

  PRINT #1, CHR$(12); 'Send read steps command
  PRINT #1, CHR$(32 + StepperNumber); 'Send stepper number + bit 5 set
  HiByte = ASC(INPUT$(1, #1))'Get the High step byte
  MdByte = ASC(INPUT$(1, #1))'Get the Middle step byte
  LoByte = ASC(INPUT$(1, #1))'Get the Low step byte
  StepsTaken& = (HiByte * 65536) + (MdByte * 256) + LoByte

```

```

END SUB

```

```

SUB Initialize (StepperNumber!, StepMode, LimitLogic)

```

```

  PRINT #1, CHR$(2); ' Send Initialize command
  'Send the combined stepper # and initialize flags
  FlagStates = StepMode + (LimitLogic * 128)
  PRINT #1, CHR$(StepperNumber! + FlagStates);

```

```

END SUB

```

```

SUB MoveCursor

```

```

  'Moves Cursor to command prompt line
  LOCATE 15, 1
  PRINT STRING$(78, " ");
  LOCATE 15, 23

```

```

END SUB

```

```

SUB ResetNode (Node, ResetResponse)

```

```

  PRINT #1, CHR$(1); 'Send reset command
  PRINT #1, CHR$(Node!); 'Send node address
  ResetResponse = ASC(INPUT$(1, #1)) 'Get reset response

```

```

END SUB

```

```

SUB RunStepper (StepperNumber, AutoRun)

    PRINT #1, CHR$(8); 'Send run command
    PRINT #1, CHR$(StepperNumber + (AutoRun * 128)); 'Send stepper number

END SUB

SUB SetAccelDecel (StepperNumber, RFlags)

    PRINT #1, CHR$(13); 'Send set ramp parameters command
    PRINT #1, CHR$(32 + StepperNumber); 'Send stepper number + bit 5 set
    PRINT #1, CHR$(RFlags); 'Send ramp select flags
END SUB

SUB SetDirection (StepperNumber, DIR)

    PRINT #1, CHR$(7); 'Send set direction command
    'Send the combined stepper # and direction flag
    PRINT #1, CHR$(StepperNumber + (DIR * 128));

END SUB

SUB SetDivisor (StepperNumber, Divisor%)

    PRINT #1, CHR$(5); 'Send set speed divisor command
    'Convert Divisor entry to Hi and Lo divisor bytes
    LoDivisor% = Divisor% MOD 256
    Divisor% = Divisor% - LoDivisor%
    HiDivisor% = Divisor% / 256
    'Send the combined stepper # and Hi divisor value
    PRINT #1, CHR$(StepperNumber + (HiDivisor% * 64));
    PRINT #1, CHR$(LoDivisor%); 'Send Lo divisor value

END SUB

SUB SetOffset (StepperNumber, RampOffset)

    PRINT #1, CHR$(13); 'Send set ramp parameters command
    PRINT #1, CHR$(64 + StepperNumber); 'Send stepper number + bit 6 set
    PRINT #1, CHR$(RampOffset); 'Send ramp offset value

END SUB

SUB SetRamp (StepperNumber, RampRate)

    PRINT #1, CHR$(13); 'Send set ramp parameters command
    PRINT #1, CHR$(StepperNumber); 'Send stepper number
    PRINT #1, CHR$(RampRate); 'Send ramp rate value

END SUB

SUB SetRampStretch (StepperNumber, SFlags)

    PRINT #1, CHR$(13); 'Send set ramp parameters command
    PRINT #1, CHR$(128 + StepperNumber); 'Send stepper number + bit 7 set
    PRINT #1, CHR$(SFlags); 'Send ramp stretch flags

END SUB

```

SUB SetSpeed (StepperNumber, Speed)

```
PRINT #1, CHR$(4); 'Send set stepper speed command
PRINT #1, CHR$(StepperNumber); 'Send stepper number
PRINT #1, CHR$(Speed); 'Send Speed value
```

END SUB

SUB SetStepCount (StepperNumber, StepCount&)

```
'Convert the step count into High, Medium and Low bytes
LoSteps = StepCount& MOD 256
StepCount& = StepCount& - LoSteps
StepCount& = StepCount& / 256
MdSteps = StepCount& MOD 256
StepCount& = StepCount& - MdSteps
HiSteps = StepCount& / 256
PRINT #1, CHR$(6); 'Send the set step count command
'Send the combined stepper # and HiSteps byte
PRINT #1, CHR$(StepperNumber + (HiSteps * 64));
'Send the MdSteps byte
PRINT #1, CHR$(MdSteps);
'Send the LoSteps byte
PRINT #1, CHR$(LoSteps);
```

END SUB

SUB StartStepper (StepperNumber, AutoStart)

```
PRINT #1, CHR$(9); 'Send start command
PRINT #1, CHR$(StepperNumber + (AutoStart * 128)); 'Send stepper number
```

END SUB

SUB StopStepper (StepperNumber)

```
PRINT #1, CHR$(10); 'Send stop command
PRINT #1, CHR$(StepperNumber); 'Send stepper number
```

END SUB