

Eggert Electronics Engineering

Daniel N. Eggert • 125 Pasa Por Aqui Ln. • Alamogordo, NM 88310 • Tel: 575-437-0698

The SSC3 Stepper Motor Control System

(For Command Interpreter Versions 3.0 and above)

The SSC3 interface circuit board can control up to three 4-phase unipolar stepper motors simultaneously and independently with commands sent from a personal computers serial port. Stepper motor motion is performed by the SSC3 interface in the foreground while it receives commands from your PC in the background. All three stepper motors are independent of each other. While one stepper motor is in motion you can be setting up step counts and direction for another stepper motor. Your PC can be busy computing the position coordinates for the next move while the SSC3 interface is busy positioning motors from previously sent commands.

The SSC3 interface circuit board design is simple and straightforward. It includes an RS-232 interface, several buffered general-purpose digital inputs and outputs, and 12 power drive transistor circuits for the stepper motors. The hardware that you will need to control the motion of three stepper motors is the SSC3 interface circuit board, a serial interface cable to connect it to your PC, and power sources for the interface and stepper motors. Example programs are provided to help you get started with your CNC (Computer Numeric Control) and Robotics projects. The stepper motor exerciser programs are written in QuickBasic and Visual Basic for Windows. A complete listing of the QuickBasic stepper motor exerciser program (QBSC3EXR.BAS) is provided after the command summary section. Figure 1 illustrates how the various components of the SSC3 system are interconnected to the interface circuit board. Only one of the stepper motors is shown connected in this illustration.

The SSC3 system interprets fourteen different commands. The first three commands initialize the interface for full or half step mode, 1 or 2 phase excitation mode, and mechanical limit sense. Nine of the commands control stepper motor motion. One command controls the logic states of eleven digital outputs. Seven of the digital outputs are for general-purpose use. The other four digital outputs are optionally selected at system initialization for either stepper motor three drive outputs or high-powered open collector general-purpose digital outputs. A command reads the logic states of eleven digital inputs. Five of the digital inputs are for general-purpose use, and the other six digital inputs can be optionally selected at system initialization for either stepper motor mechanical limit sense inputs or general-purpose digital inputs. If they are initialized as stepper motor mechanical limit sense inputs then the stepper motor will stop immediately whenever a limit is reached. If limit switch sense inputs are not needed for a stepper motor then the inputs can be used for various other things.

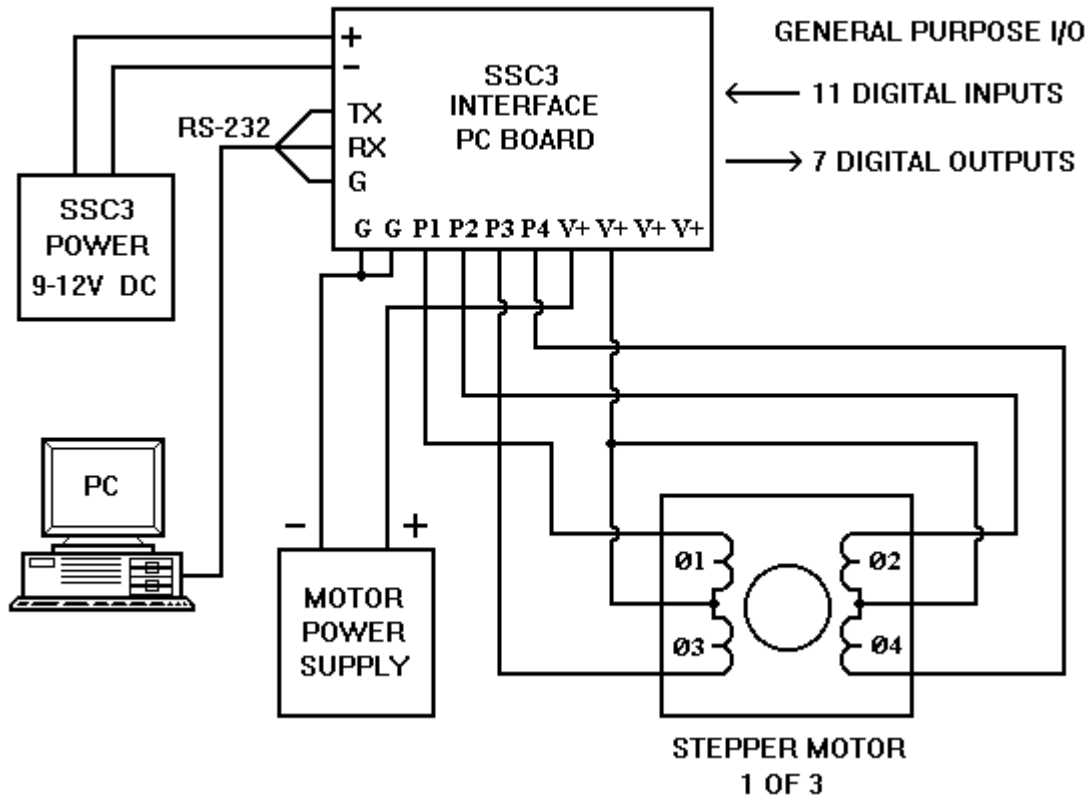


Figure 1. SSC3 System Interconnect

About The Circuit

The SSC3 printed circuit board schematic diagram is located at the back of this manual. The heart of the circuit is the 8C751 embedded controller IC1. All four output ports of the microcontroller are buffered with IC2 thru IC5. These octal buffer and line driver IC's are used to protect the microcontrollers I/O lines and provide adequate output levels. JP1 is a 2 pin header on the circuit board that can be momentarily shorted for resetting the interface. The RS-232 serial port circuitry at IC6 utilizes a MAX232, which has built-in charge pump voltage converters and RS-232 receivers and transmitters. The embedded controller IC1 has a built-in serial interface with a programmable baud rate generator.

A frequency of 11.0592 MHz was chosen for crystal Y1 so that IC1 can generate the SSC3 systems **FIXED COMMUNICATIONS RATE OF 9600 BAUD, NO PARITY, 8 DATA BITS AND 1 STOP BIT.**

All of the SSC3 interface digital inputs are pulled up to +5 volts by the 10K ohm resistors in the resistor network RN1 and resistors R2, R3 and R4. The SSC3 interface digital inputs are connected at terminal strip TS1 pins I1 thru I7 and terminal strip TS2 pins I8 thru I11. Their common ground connection is located at terminal strip TS1 and TS2 pins G. The optional mechanical limit sense inputs are digital inputs I1 through I6. All of the circuit board terminal connections are labeled on the top of the printed circuit board and on the schematic diagram. A normally closed switch contact connected to a digital input and ground is all that you need for a mechanical limit sense switch input.

The recommended power input for the SSC3 interface is from 9 to 15 volts DC. Typical circuit board power consumption is about 20 milli-amps with a 12 volt DC input. This power source can be from a common 9 volt DC power adapter. The voltage regulator VR1 provides the 5 volts for all of the logic and serial interface circuitry on the SSC3 interface circuit board. Diode D2 in the power input circuit protects the components from damage due to accidental polarity reversal.

The SSC3 interface digital outputs are present at terminal strip TS6 pins O1 through O4 and terminal strip TS2 pins O5, O6 and O7. Their common ground connection is at terminal strip TS6 and TS2 pins G. These digital outputs are TTL level and can directly drive the inputs of solid state relays to switch on and off 110 volt devices like drill motors or solenoids.

Only one of the twelve stepper motor drive output circuits is shown in detail on the schematic diagram. All twelve of the output circuits are identical. Diodes D3 thru D14 are no longer used in the circuit. These were the clamp diodes used for inductive load transient suppression. They are no longer needed because the IRL540N transistor that is currently used in the drive circuit has its own internal clamp diode across the source-drain junction. Pin C of TS4, TS5, and TS6 are no longer used. The terminals on TS4, TS5, and TS6 that are labeled P1, P2, P3 and P4 are the four phase motor drive outputs for the three stepper motors. Please note that **the P1 through P4 drive outputs on terminal strip TS6 are in the reverse order** for stepper motor three. The IRL540N HEXFET drive transistors Q1 thru Q12 installed on the circuit board will provide adequate drive output switching for a wide variety of small to medium stepper motors. These drive transistors can provide up to 3.0 amps continuous current per phase with the heat sinks supplied. The IRL540N transistors are rated at a maximum current of 27 amps and a maximum of 100 volts. The relatively small transistor heat sinks that are provided limit the amount of heat that the IRL540N HEXFET transistors can dissipate.

Power Supply Options

You can use figure 1 as a wiring guide when connecting the stepper motors and their power supplies to the SSC3 interface board. If wired as illustrated in figure 1 you can use any of the three stepper drive output sequence modes. However, motor torque will only be sufficient at slow speeds. For good torque at higher speeds the SSC3 drive circuitry should be used with current limiting ballast resistors in series with the motor power supply positive output and the V+ terminal on the SSC3 interface circuit board. This type of stepper motor drive circuitry is referred to as an **L/R** drive configuration. It is a low cost solution for controlling stepper motors. If the power supply for the stepper motor is fixed at the motors rated voltage as in figure 1 then the current rise time in the motor coils will be slow. As a result, the motor torque will be good at slow step rates but it will decay as the step rate increases. You can increase the motor coils current rise time by increasing the supply voltage and adding current limiting power resistors to the circuit.

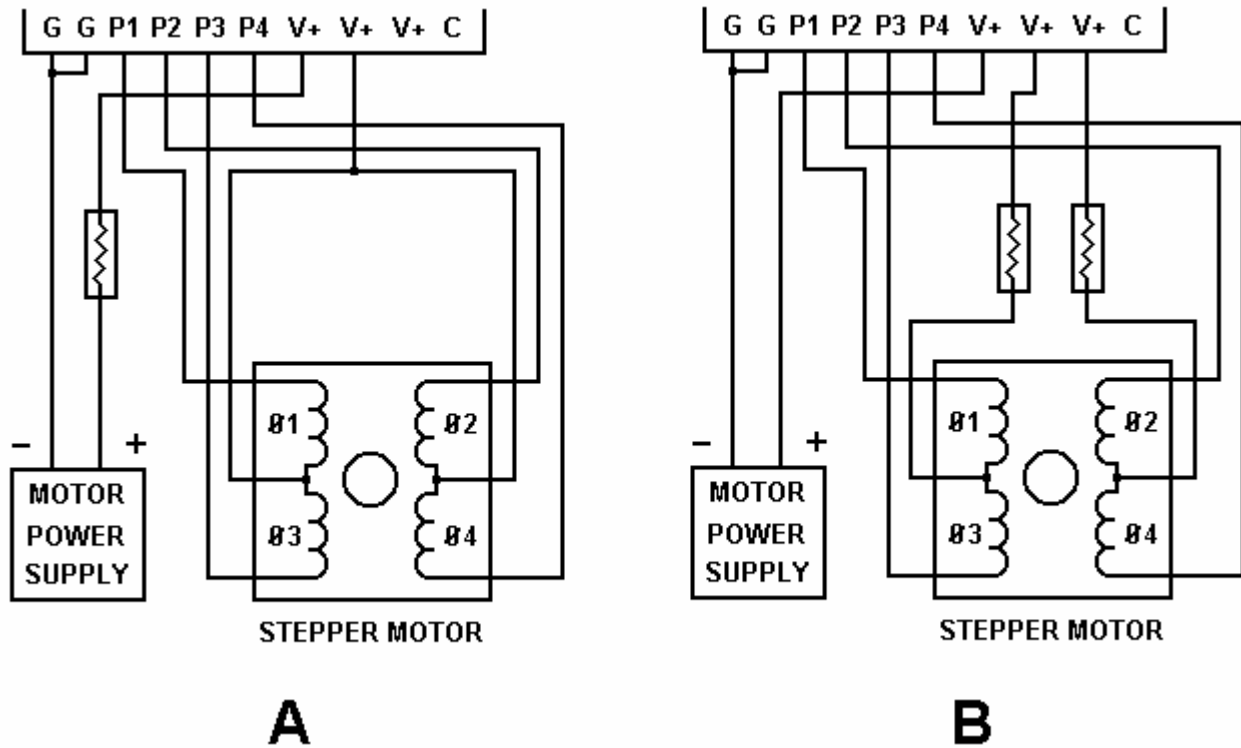


Figure 2A and 2B. Power Supply Wiring Options

A motor power supply voltage that is 2 to 4 times the motors voltage rating is recommended. You could use one common power supply for all three stepper motors and the interface circuit board if you use wiring combinations of figures 2A and 2B. As an example, two motors that move an X and Y positioning table are rated 5.5 volts at 1.3 amps per phase. The stepper motor that raises and lowers the drill press above the table is rated 5.0 volts at 450 milli-amps per phase. A power resistor is placed between the positive lead of a 12 volt power supply and a V+ terminal of each of the three stepper motor drive output terminals. The resistance values and power ratings of these resistors are determined using ohms law. The power resistor value required for use with the 5.5 volt 1.3 amp motors can be found by first subtracting the motors rated voltage from the motors power supply voltage ($12.0V - 5.5V = 6.5V$). The motor coil will have 5.5 volts drop across it with the remaining 6.5 volts of the motor power supply dropped across the current limiting resistor. Using ohms law ($R=E/I$) ($5.0=6.5/1.3$), the power resistor will have a value of 5.0 ohms. The wattage of the power resistor is calculated using ohms law ($P=I*E$) ($8.45=1.3*6.5$). It is good practice to select a power resistor with at least twice the power rating that was calculated. A 20 watt 5.0 ohm power resistor is recommended. Some cooling air circulation may be required! A 11.0 ohm 10 watt resistor would be used for the stepper motor that raises and lowers the drill press. If you use 1 current limiting resistor as illustrated in figure 2A you will be limited to one or two phase excitation modes only. Also, if you use two phase excitation mode with the arrangement of figure 2A then a power resistor of half the resistance value and twice the power rating has to be used. This is due to the fact that there is always two phases energized when in two phase excitation mode. The voltage drop across the power

resistor is the same, but twice the current will flow through it. The best configuration is illustrated in figure 2B. This arrangement uses a series current limiting resistor for each stepper motor coil. You will be able to use any of the 3 modes with this configuration. Half step mode alternately has one and then two phases energized while stepping and therefore requires the use of current limiting resistors for each motor coil.

Serial interface to PC

The RS-232 serial interface connection from the SSC3 interface circuit board 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. The shield should be connected at connector pin 1 if it is used. Do not connect the shield to anything at the SSC3 interface end of the cable. A 9 foot unshielded RS-232 cable assembly with a 9-pin connector is supplied with the SSC3 system. It has 3 tinned wires at the end of the cable that connects to terminal strip TS3 on the SSC3 interface printed 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-pin or 25-pin serial port connector. You may have to use a 9-pin to 25-pin adapter with the cable that is supplied with the SSC3 system.

Use figure 3 as a guide when fabricating your own cable assembly. Be sure to include the two jumper wires on the connector. The GWBASIC and QuickBASIC example programs provided with the SSC3 system will not work properly without these jumpers. The jumpers loop back the handshaking signals. The wire jumpers force your computers serial communications hardware to think that it is handshaking with the SSC3 interface serial port. These jumpers can be omitted only if the DS0, CS0, CD0 and RS options are added to the OPEN COM statement in your GWBASIC and QuickBASIC programs. These options in the OPEN COM statement tell the serial communications hardware to ignore the hardware handshaking signals.

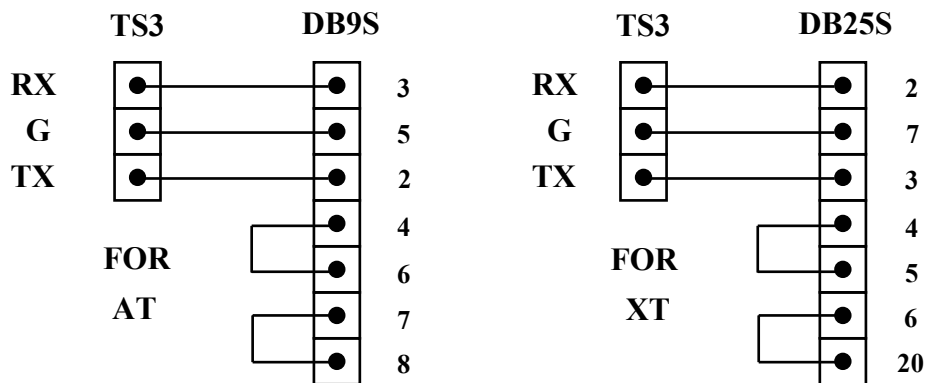
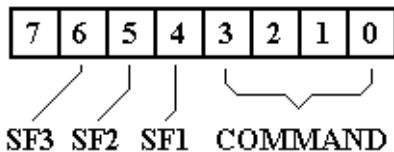


Figure 3. Serial Cable Wiring Diagrams

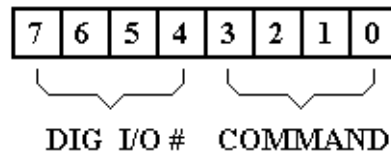
The SSC3 Command Summary

There are fourteen commands in the SSC3 system. The lower four bits of the command byte (bits 0 through 3) contain the actual command. Most of the commands use Bits 4, 5 and 6 of the command byte as flags corresponding to stepper motors 1, 2 and 3 respectively. Some commands like the start and stop commands only take action if the stepper flag is set. Other commands like the change direction and auto drive enable commands take action with both possible logic states of the flags. The upper nibble of the command byte (bits 4 through 7) is used to indicate the digital I/O number for the read digital input and change digital output commands. Commands 1, 2 and 3 are initialization commands. You must initialize the SSC3 interface after a reset or power up by sending commands 1, 2 and 3. After initialization has been performed, the initialization parameters cannot be changed again until after an interface power-up or reset occurs. No digital outputs or stepper motor drive outputs will be enabled until initialization is performed. The 3 initialization commands can be sent in any order.

COMMAND WORD FORMAT
FOR COMMANDS
1, 2, 3, 5, 6, 7, 8, 9, 10, 11, 14



COMMAND WORD FORMAT
FOR COMMANDS 12 AND 13



Command 1 ... Initialize for full or half step

Full or half step mode is initialized for all three stepper motors simultaneously with this command. Bits 4, 5 and 6 of this command byte are the stepper flags. A flag is cleared for full step mode and set for half step mode. See table 1 for the stepper motor drive output logic states for the three modes of operation.

The S in the step direction columns on the left side of table 1 indicates the stepper motor output states after system initialization takes place. The stepper motor could also be stopped at any one of the step positions 1 through 8 if it has been commanded to move since the system was initialized. The rows to the right of each of the steps in the **CW** and **CCW** columns indicate the output logic states of the P1, P2, P3, and P4 terminals on the SSC3 interface circuit board for all three modes. A black square indicates that the stepper motor coil is energized. All three of the stepper motors can use different modes.

STEP		FULL STEP MODES								HALF STEP			
DIRECTION		1 PHASE EXC				2 PHASE EXC							
CW	CCW	P1	P2	P3	P4	P1	P2	P3	P4	P1	P2	P3	P4
8	8	■				■	■			■			
1	7		■				■	■		■	■		
2	6			■				■	■		■		
3	5				■	■			■		■	■	
4	4	■				■	■					■	
5	3		■				■	■				■	■
6	2			■				■	■				■
7	1				■	■			■	■			■
8	8	■				■	■			■			

Table 1. Stepper Motor Drive Output Logic States

Command 2 ... Initialize stepper limit sense/Outputs 8-11 select

Initialize stepper limit sense

Limit sense is initialized for all three stepper motors simultaneously with this command. Bits 4, 5 and 6 of this command byte are the stepper flags. A flag is cleared if limit sense is not used and set if limit sense is used. The mechanical limit sense inputs are digital inputs I1 through I6. They are labeled on the schematic diagram as CCW1 and CW1 for stepper 1, CCW2 and CW2 for stepper 2, CCW3 and CW3 for stepper 3. CCW stands for the counter-clockwise limit and CW stands for the clockwise limit. The CCW limit switch will only be detected when the motors output shaft is turning counter-clockwise. The CW limit switch will only be detected when the motors output shaft is turning clockwise. Limit sense will not work if the limit switches are reversed. Your stepper motor will not rotate if you have initialized mechanical limit sense with this command and you do not have a normally closed switch contact or jumper connected between the limit switch inputs and ground. The state of these six digital inputs can always be read with the read digital input command 12. When limit sense is initialized for a stepper motor the SSC3 interface will immediately stop stepper motion when a limit input goes to a high logic state. The stepper motor can be moved off of the mechanical limit by stepping it in the opposite direction. All digital inputs are pulled high with a 10K ohm resistor. A normally closed switch between the input terminal and ground is all that is needed.

Outputs 8-11 select

Bit 7 in command 2 is now used with versions 3.0 and above of the SSC3 system command interpreter firmware so that the 4 drive outputs of stepper motor 3 can be selected for use as general purpose digital outputs 8, 9, 10 and 11. When your application does not need 3 stepper motors for motion control then the drive outputs of stepper 3 can be used as high powered open collector digital outputs for energizing relay coils, solenoids and various other DC powered devices. If bit 7 in command 2 is set during system initialization then the 4 drive outputs at TS6 pins P1 through P4 will become general purpose digital outputs 8 through 11. The command interpreter will ignore any commands that would normally control stepper motor 3 when this option **is** selected. If this option is **not** selected then any output state changes that are sent for digital outputs 8 through 11 with command 13 (change digital output state) will not be recognized.

Command 3 ... Initialize for 1 or 2 phase excitation

1 or 2 phase excitation mode is initialized for all three stepper motors simultaneously with this command. Bits 4, 5 and 6 of this command byte are the stepper flags. A flag is cleared for 1-phase excitation mode and set for 2-phase excitation mode. The state of the stepper flag in this command is ignored if a stepper motor is initialized for half step operation with command 1. The 1 and 2 phase excitation modes are sometimes referred to as monophasic and biphasic modes. See table 1 above for the stepper motor drive logic output states for these modes.

Command 4 ... Change the step rate clock

This command changes the step rate clock. The step rate clock is the time base generator for the stepper motors. Changing the step rate clock affects the speed of all three stepper motors. The SSC3 interface will ignore the step rate clock command if any of the steppers are in motion when the command is received. The step rate clock range setting is from 1 to 240 decimal. Table 2 below indicates the most popular step rate clock settings and frequencies. A default step rate clock setting of 199 (502 Hz) is selected when the SSC3 interface is initialized at power up or reset. The slowest step rate clock setting is 1. A value of 1 is set if a step rate clock setting of 0 is received. A value of 240 is set if a value larger than 240 is received. There are no flags associated with this command. The upper nibble in the command byte is not used. The step rate clock byte must be sent immediately following this command.

SETTING	STEPS-PER-SECOND
15	120
63	150
112	200
136	240
141	250
160	300
176	360
184	400
199	502
228	1012
240	1748

Table 2. Step Rate Clock Settings

Command 5 ... Load the speed divisor(s)

This command allows you to change any one or more stepper motor speeds by a given step rate clock division. The speed divisor range is from 1 to 255. The SSC3 interface will set the speed divisor to 1 if it receives a speed divisor of 0. A speed divisor setting of 1 will cause the step rate to be equal to the current setting of the step rate clock. A speed divisor setting of 2 will cause a step rate of one half the speed of the step rate clock setting and so on, up to a maximum division of 255. A default speed divisor of 10 is set for all steppers when the SSC3 interface is initialized at power up or reset. Speed divisors can be changed while the stepper motors are in motion. Bits 4, 5 and 6 of this command byte are the stepper flags. The flag is set if the stepper is to receive the speed divisor. The speed divisor byte must be sent immediately after this command.

Command 6 ... Load the step count

This command loads the step count for any one or more stepper motors. The step count is a 16-bit word with a range of 1 to 65535. Bits 4, 5 and 6 of this command byte are the stepper flags. The flag is set if the stepper is to receive the step count. The two-byte step count must be sent immediately after this command. The low byte of the step count is sent first followed by the high byte. The step count will not be changed if the stepper motor is in motion when the command is received by the SSC3 interface.

Command 7 ... Change the stepper(s) direction

This command simultaneously sets the direction for all three of the stepper motors. A default direction of clockwise is set when the SSC3 interface is initialized at power up or reset. The direction will not be changed if the stepper motor is in motion when the SSC3 interface receives the command. Bits 4, 5 and 6 of this command

byte are the stepper flags. The flag is cleared if you want the stepper direction to be clockwise or set for counterclockwise. Motor direction is determined when the end of the stepper motor output shaft is facing toward you.

Command 8 ... Start stepper(s)/Run stepper(s)

Start stepper(s)

This command starts moving any one or more steppers. Bits 4, 5 and 6 of this command byte are the stepper flags. If a flag is set and there are step counts remaining then the stepper will start moving after the execution of this command.

Run stepper(s)

Bit 7 in command 8 is now used with versions 3.0 and above of the SSC3 system command interpreter firmware for the continuous run mode. If bit 7 in command 8 is set then the stepper(s) will start stepping and will not stop until the stop stepper(s) command is sent. If limit sense has been initialized for a stepper motor then it will also stop when a mechanical limit is reached. A step count does not have to be loaded before a continuous run command is sent. When the command interpreter finds bit 7 set in the command word it forces the step count for the stepper(s) to 255 and does not decrement the step count while the stepper motor is in motion. The stop stepper(s) command 9 resets the continuous run flag for the stepper(s).

Command 9 ... Stop stepper(s)

This command will stop the motion of any one or more steppers. Bits 4, 5 and 6 of this command byte are the stepper flags. If a flag is set then the stepper motor will stop after the execution of this command. The motor will stop immediately regardless of the ramp rate set with command 11 and the drive outputs will turn off immediately if auto drive was enabled with command 14. This command may never be needed in an application because a stepper motor stops automatically whenever it has finished stepping the number of steps loaded by command 6, or when limit sense has been initialized for a stepper motor and a limit is reached. A digital input would be used in conjunction with this command in a control panel manual emergency stop button software routine.

Command 10 ... Get stepper status/Read step count

Get stepper status

This command fetches a stepper status byte from the SSC3 interface. Bits 4, 5 and 6 of this command byte are the stepper flags. After the execution of this command a status byte for the indicated stepper motor is sent back to the host PC. Only one stepper flag is set in the command byte, and the status byte that is received only indicates the status for one stepper motor. If more than one stepper flag is set in the command byte then the SSC3 interface scans bits 4, 5 and 6 and sends a status byte for the first stepper flag that it finds set. If none of the stepper flags are set then it sends a status byte for stepper 1. Bit 0 of the status byte is the position complete flag. If the stepper motor has finished stepping the desired amount

of step counts set by command 6 then the position complete flag in the status byte will be set. Bit 4 in the status byte is the at-limit flag. If mechanical limit sense was initialized and the stepper motor has reached a clockwise or counter-clockwise limit then this flag will be set. The status byte does not indicate which limit the stepper has reached. You can read the states of the clockwise and counter-clockwise limits with command 12 to determine which limit has been reached.

Read step count

If bit 7 in command 10 is set then the SSC3 command interpreter will send back the current low and high step count bytes respectively for the stepper motor that is selected with bits 4, 5 and 6. As in the Read Status command, this command sends the step count for only one stepper motor. The first flag that it finds set in bits 4, 5 and 6 respectively will be the motor that is selected for this command. This is a new feature that is incorporated with the SSC3 system command interpreter firmware versions 3.0 and above. You can re-combine the low and high step count bytes to form a 16-bit step count with the following statement: **[StepCount = (HiByte * 256) + LoByte]**.

Command 11 ... Select the stepper(s) ramp rate

This command loads a ramp rate value for any one or more of the steppers. The ramp rate range is from 0 to 255 decimal. A default ramp rate of 0 is set when the SSC3 interface is initialized at power up or reset. Bits 4, 5 and 6 of this command byte are the stepper flags. The ramp rate value must be sent immediately after this command. The ramp rate value modifies the steppers speed divisor and is non-linear for this reason. It modifies the speed divisor for up to 255 of the first and last steps of the step count. It provides a means of ramping the acceleration and deceleration of the steppers speed. It seems to work best when the stepper speed divisor is set at a value of 7 or larger and the ramp rate is 150 or larger. Some experimentation is needed for optimization with your particular application.

Command 12 ... Read digital input state

This command reads the logic state of a digital input. There are eleven digital inputs. The upper nibble (bits 4 through 7) of the command byte contains the digital input number to be read. The status byte indicating the logic state of the digital input is sent back by the SSC3 interface immediately after the command is sent. The status byte is a decimal 0 or 1. The logic state of digital inputs 1 through 6 can be read even if they were initialized as stepper motor mechanical limit sense inputs.

Command 13 ... Change digital output state

This command changes the logic state of a digital output. There are 7 general-purpose digital outputs in the SSC3 system. High-powered digital outputs 8 through 11 can be optionally selected with command 2 during system initialization instead of stepper motor three drive outputs. Bits 4, 5 and 6 in this command contain the digital output number. The new output state is a decimal 0 or 1 and must be sent immediately after this command.

Command 14 ... Enable/disable auto drive output mode.

This command allows you to enable or disable auto drive output mode. Bits 4, 5 and 6 of this command byte are the stepper flags. If they are set then auto drive output mode is enabled after the command is sent. If they are cleared then auto drive output will be disabled. All stepper motor auto drive output modes are simultaneously affected by this command. These flags are cleared at system reset. If auto drive output mode is enabled then all of the four phase motor drive outputs will turn off when the stepper motor is stopped. Turning off a stepper motors drive outputs when it is stopped will prevent unnecessary current drain on the power supply and overheating of the stepper motor. On the other hand, the stepper motor drive outputs may need to be enabled in order to provide locking torque that will hold the stepper motor at its rest position. Locking torque may not be necessary in situations where the steps-per-inch ratio of the positioning system is very high or when it is not important to maintain an absolute rest position. The motor drive outputs will turn on one step time before the stepper motor is sent into motion and they will remain on for one step time after the motor has finished stepping. The stepper motors position complete flag in the status byte fetched with command 10 will be set the instant the motor drive outputs are turned off. If auto drive output mode is disabled then the stepper motors drive outputs will never be turned off. You can enable or disable auto drive output mode at any time. Enabling or disabling auto drive output mode for a stepper motor when it is stopped will turn on or off the motor drive outputs instantaneously. Enabling or disabling auto drive output mode when a stepper motor is in motion will have no effect until it stops. You can enable auto drive output mode before you finish initialize the system with commands 1, 2 and 3 if you want the stepper motor drive outputs to be off after the system is initialized.

System reset

The SSC3 interface will be reset if the 2-byte sequence of a decimal 80 and a decimal 95 is sent as a command. A decimal 5 is sent back by the SSC3 interface to confirm the reset.

Communications sync

Versions 3.0 and above of the SSC3 system command interpreter firmware allows you to send a communications sync command which can be used to test serial communications integrity. The sync command (decimal 96) is sent followed by a sync byte (any 8 bit value). After the SSC3 command interpreter receives the sync command and sync byte it will send back the sync bytes upper nibble decimal value to the host PC. The sync byte that the SSC3 command interpreter sends back to the host PC is not an exact copy of the byte sent by the host PC, but it is a decimal value between 0 and 15 that can be predicted by the host PC's application program.

Initialization basics

The SSC3 interface will accept and execute any of the commands before system initialization has been completed. However, motor drive outputs and digital outputs will be at logic low states until system initialization has been completed. You can set up various parameters like the speed divisors, auto drive enable mode and load step counts before you send commands 1, 2 and 3 for final initialization.

Program example

The example QuickBasic program (QBSC3EXR.BAS) of Listing 1 below is a stepper motor exerciser program that utilizes all of the SSC3 system commands. It is intended to illustrate how easy it is to command and control three stepper motors with the system. A stepper motor exerciser program written in Visual Basic 6.0 is also included on the SSC3 system programming examples disk.

' Listing 1. QBSC3EXR.BAS, A Simple Stepper Motor Exerciser Program

```
DECLARE SUB ClearText ()
DECLARE SUB GetStepsRemaining (SF3!, SF2!, SF1!, StepsRemaining&)
DECLARE SUB RunStepper (SF3!, SF2!, SF1!)
DECLARE SUB InitializeMode (HalfStep3!, HalfStep2!, HalfStep1!)
DECLARE SUB InitializeLimits (DO811!, Limits3!, Limits2!, Limits1!)
DECLARE SUB InitializePhases (Phases3!, Phases2!, Phases1!)
DECLARE SUB SetStepRate (rate!)
DECLARE SUB SetDivisor (SF3!, SF2!, SF1!, SPEED!)
DECLARE SUB SetStepCount (SF3!, SF2!, SF1!, Steps&)
DECLARE SUB SetDirection (Dir3!, Dir2!, Dir1!)
DECLARE SUB StartStepper (SF3!, SF2!, SF1!)
DECLARE SUB StopStepper (SF3!, SF2!, SF1!)
DECLARE SUB GetStatus (SF3!, SF2!, SF1!, ReceivedStatus!)
DECLARE SUB SetRamp (SF3!, SF2!, SF1!, RRATE!)
DECLARE SUB GetInput (DINBR!, InputValue!)
DECLARE SUB ChangeOutput (DONBR!, STATE!)
DECLARE SUB ChangeAuto (Auto3!, Auto2!, Auto1!)
DECLARE SUB ResetInterface (ResetResponse!)
DECLARE SUB MoveCursor ()

' ***** QUICK BASIC STEPPER MOTOR *****
' ***** EXERCISER PROGRAM FOR THE *****
' ***** SSC3 STEPPER MOTOR *****
' ***** CONTROL SYSTEM *****
' ***** 05/09/02 *****

' Daniel N. Eggert
' 125 Pasa Por Aqui Ln.
' Alamogordo, NM 88310
' Phone (505) 437-0698

SF1 = 1: SF2 = 0: SF3 = 0: InitFlag = 0

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

' Do not start until the input buffer is empty!

DO UNTIL EOF(1)
  I = ASC(INPUT$(1, #1))
LOOP

PRINT
PRINT " STEPPER EXERCISER MENU"
PRINT
PRINT " 1 - Reset Interface 9 - Run Stepper(s)"
PRINT " 2 - Initialize Interface 10 - Stop Stepper(s)"
```

```

PRINT "      3 - Change Stepper Flag(s)          11 - Read Stepper Status"
PRINT "      4 - Change Step Rate Clock        12 - Read Steps Remaining"
PRINT "      5 - Change Speed Divisor(s)        13 - Change Ramp Rate(s) "
PRINT "      6 - Load Step Count(s)            14 - Read Digital Input"
PRINT "      7 - Change Direction(s)            15 - Change Digital Output"
PRINT "      8 - Start Stepper(s)                16 - Change Auto Drive(s) "
PRINT

```

```
'Display stepper preset variables
```

```

LOCATE 21, 10: PRINT "DIV = 10"; : LOCATE 21, 35: PRINT "DIV = 10";
LOCATE 21, 60: PRINT "DIV = 10";
LOCATE 22, 10: PRINT "DIR = CW"; : LOCATE 22, 35: PRINT "DIR = CW";
LOCATE 22, 60: PRINT "DIR = CW";
LOCATE 16, 60: PRINT "STEP RATE = 199";
LOCATE 18, 5: PRINT "STEPPER #1 VARIABLES";
LOCATE 18, 30: PRINT "STEPPER #2 VARIABLES";
LOCATE 18, 55: PRINT "STEPPER #3 VARIABLES";
LOCATE 20, 8: PRINT "FLAG STATE = 1";
LOCATE 20, 33: PRINT "FLAG STATE = 0";
LOCATE 20, 58: PRINT "FLAG STATE = 0";

```

```
DO
```

```

CALL MoveCursor
INPUT "Enter Menu Number "; MenuNumber

```

```
SELECT CASE MenuNumber
```

```
  CASE 1 'Reset the SSC3 Interface
```

```

    CALL ClearText
    CALL ResetInterface(ResetResponse)
    IF ResetResponse = 5 THEN
      LOCATE 16, 5
      PRINT "RESET CONFIRMED";
      InitFlag = 0
    END IF

```

```
  CASE 2 'Initialize the SSC3 Interface
```

```

    CALL ClearText
    IF InitFlag = 0 THEN
      CALL MoveCursor
      INPUT "Select half step mode for stepper #1 Y/n "; A$
      IF A$ = "n" OR A$ = "N" THEN HalfStep1 = 0 ELSE HalfStep1 = 1
      CALL MoveCursor
      INPUT "Select limits for stepper #1 Y/n "; A$
      IF A$ = "n" OR A$ = "N" THEN Limits1 = 0 ELSE Limits1 = 1
      CALL MoveCursor
      INPUT "Select 2 phase excitation for stepper #1 Y/n "; A$
      IF A$ = "n" OR A$ = "N" THEN Phases1 = 0 ELSE Phases1 = 1
      CALL MoveCursor
      INPUT "Select half step mode for stepper #2 Y/n "; A$
      IF A$ = "n" OR A$ = "N" THEN HalfStep2 = 0 ELSE HalfStep2 = 1
      CALL MoveCursor
      INPUT "Select limits for stepper #2 Y/n "; A$
      IF A$ = "n" OR A$ = "N" THEN Limits2 = 0 ELSE Limits2 = 1
      CALL MoveCursor
    END IF

```

```

INPUT "Select 2 phase excitation for stepper #2 Y/n "; A$
IF A$ = "n" OR A$ = "N" THEN Phases2 = 0 ELSE Phases2 = 1
CALL MoveCursor
INPUT "Select half step mode for stepper #3 Y/n "; A$
IF A$ = "n" OR A$ = "N" THEN HalfStep3 = 0 ELSE HalfStep3 = 1
CALL MoveCursor
INPUT "Select limits for stepper #3 Y/n "; A$
IF A$ = "n" OR A$ = "N" THEN Limits3 = 0 ELSE Limits3 = 1
CALL MoveCursor
INPUT "Select 2 phase excitation for stepper #3 Y/n "; A$
IF A$ = "n" OR A$ = "N" THEN Phases3 = 0 ELSE Phases3 = 1
CALL MoveCursor
INPUT "Use Digital Outputs 8-11 at Stepper 3 Outputs Y/n "; A$
IF A$ = "n" OR A$ = "N" THEN DO811 = 0 ELSE DO811 = 1
CALL MoveCursor
CALL InitializeMode(HalfStep3, HalfStep2, HalfStep1)
CALL InitializeLimits(DO811, Limits3, Limits2, Limits1)
CALL InitializePhases(Phases3, Phases2, Phases1)
InitFlag = 1
ELSE
  LOCATE 16, 5
  PRINT "Command Aborted, Already Initialized!";
END IF

CASE 3 'Change the Stepper Flag States
CALL ClearText
CALL MoveCursor
INPUT "Set or Clear flag state for stepper 1 S/C "; A$
IF A$ = "s" OR A$ = "S" THEN
  SF1 = 1: LOCATE 20, 8: PRINT "FLAG STATE = 1";
ELSE
  SF1 = 0: LOCATE 20, 8: PRINT "FLAG STATE = 0";
END IF
CALL MoveCursor
INPUT "Set or Clear flag state for stepper 2 S/C "; A$
IF A$ = "s" OR A$ = "S" THEN
  SF2 = 1: LOCATE 20, 33: PRINT "FLAG STATE = 1";
ELSE
  SF2 = 0: LOCATE 20, 33: PRINT "FLAG STATE = 0";
END IF
CALL MoveCursor
INPUT "Set or Clear flag state for stepper 3 S/C "; A$
IF A$ = "s" OR A$ = "S" THEN
  SF3 = 1: LOCATE 20, 58: PRINT "FLAG STATE = 1";
ELSE
  SF3 = 0: LOCATE 20, 58: PRINT "FLAG STATE = 0";
END IF

CASE 4 'Change Step Rate Clock
CALL ClearText
DO
  CALL MoveCursor
  INPUT "Enter new step rate clock setting "; rate
  IF rate < 0 OR rate > 255 THEN BEEP
LOOP WHILE rate < 0 OR rate > 255

```

```

LOCATE 16, 60: PRINT "STEP RATE ="; rate; " ";
CALL SetStepRate(rate)

CASE 5 'Change Speed Divisor(S)
CALL ClearText
DO
CALL MoveCursor
INPUT "Enter speed divisor value (1-255) "; SPEED
IF SPEED < 1 OR SPEED > 255 THEN BEEP
LOOP WHILE SPEED < 1 OR SPEED > 255
IF SF1 = 1 THEN LOCATE 21, 10: PRINT "DIV ="; SPEED;
IF SF2 = 1 THEN LOCATE 21, 35: PRINT "DIV ="; SPEED;
IF SF3 = 1 THEN LOCATE 21, 60: PRINT "DIV ="; SPEED;
CALL SetDivisor(SF3, SF2, SF1, SPEED)

CASE 6 'Load Step Count(s)
CALL ClearText
DO
CALL MoveCursor
INPUT "Enter step count (1-65535) "; Steps&
IF Steps& < 0 OR Steps& > 65535 THEN BEEP
LOOP WHILE Steps& < 0 OR Steps& > 65535
CALL SetStepCount(SF3, SF2, SF1, Steps&)

CASE 7 'Change Direction(s)
CALL ClearText
DO
CALL MoveCursor
INPUT "Enter stepper #1 direction 0=CW, 1=CCW "; DIR
IF DIR < 0 OR DIR > 1 THEN BEEP
LOOP WHILE DIR < 0 OR DIR > 1
IF DIR = 1 THEN Dir1 = 1 ELSE Dir1 = 0
DO
CALL MoveCursor
INPUT "Enter stepper #2 direction 0=CW, 1=CCW "; DIR
IF DIR < 0 OR DIR > 1 THEN BEEP
LOOP WHILE DIR < 0 OR DIR > 1
IF DIR = 1 THEN Dir2 = 1 ELSE Dir2 = 0
DO
CALL MoveCursor
INPUT "Enter stepper #3 direction 0=CW, 1=CCW "; DIR
IF DIR < 0 OR DIR > 1 THEN BEEP
LOOP WHILE DIR < 0 OR DIR > 1
IF DIR = 1 THEN Dir3 = 1 ELSE Dir3 = 0
LOCATE 22, 10
IF Dir1 = 0 THEN PRINT "DIR = CW "; ELSE PRINT "DIR = CCW";
LOCATE 22, 35
IF Dir2 = 0 THEN PRINT "DIR = CW "; ELSE PRINT "DIR = CCW";
LOCATE 22, 60
IF Dir3 = 0 THEN PRINT "DIR = CW "; ELSE PRINT "DIR = CCW";
CALL SetDirection(Dir3, Dir2, Dir1)

CASE 8 'Start Stepper(s)
CALL ClearText
CALL MoveCursor

```

```

IF InitFlag = 1 THEN
  INPUT "Start stepper(s) Y/n "; A$
  IF A$ = "y" OR A$ = "Y" OR A$ = "" THEN
    CALL StartStepper(SF3, SF2, SF1)
  END IF
ELSE
  LOCATE 16, 5
  PRINT "Command Aborted, Not Initialized!";
END IF

CASE 9 'Run Stepper(s)
  CALL ClearText
  CALL MoveCursor
  IF InitFlag = 1 THEN
    INPUT "Run Stepper(s) Y/n "; A$
    IF A$ = "y" OR A$ = "Y" OR A$ = "" THEN
      CALL RunStepper(SF3, SF2, SF1)
    END IF
  ELSE
    LOCATE 16, 5
    PRINT "Command Aborted, Not Initialized!";
  END IF

CASE 10 'Stop Stepper(s)
  CALL ClearText
  CALL MoveCursor
  IF InitFlag = 1 THEN
    INPUT "Stop stepper(s) Y/n "; A$
    IF A$ = "y" OR A$ = "Y" OR A$ = "" THEN
      CALL StopStepper(SF3, SF2, SF1)
    END IF
  ELSE
    LOCATE 16, 5
    PRINT "Command Aborted, Not Initialized!";
  END IF

CASE 11 'Read Stepper Status
  CALL ClearText
  CALL GetStatus(SF3, SF2, SF1, ReceivedStatus)
  Stepper$ = "Stepper 1 "
  IF SF3 = 1 THEN Stepper$ = "Stepper 3 "
  IF SF2 = 1 THEN Stepper$ = "Stepper 2 "
  IF SF1 = 1 THEN Stepper$ = "Stepper 1 "
  IF ReceivedStatus = 1 OR ReceivedStatus = 17 THEN
    PosComp$ = "Pos. Comp. Flag = 1"
  ELSE
    PosComp$ = "Pos. Comp. Flag = 0"
  END IF
  IF ReceivedStatus >= 16 THEN
    Limit$ = "Limit Flag = 1"
  ELSE
    Limit$ = "Limit Flag = 0"
  END IF
  LOCATE 16, 5: PRINT Stepper$ + PosComp$ + ", " + Limit$;

```

```

CASE 12 'Read Steps Remaining
  CALL ClearText
  CALL GetStepsRemaining(SF3, SF2, SF1, StepsRemaining&)
  Stepper$ = "Stepper 1 "
  IF SF3 = 1 THEN Stepper$ = "Stepper 3 "
  IF SF2 = 1 THEN Stepper$ = "Stepper 2 "
  IF SF1 = 1 THEN Stepper$ = "Stepper 1 "
  LOCATE 16, 5: PRINT Stepper$ + "Steps Remaining =";
  PRINT STR$(StepsRemaining&);

CASE 13 'Change Ramp Rate(s)
  CALL ClearText
  DO
    CALL MoveCursor
    INPUT "Enter ramp rate (0-255) "; RRATE
    IF RRATE < 0 OR RRATE > 255 THEN BEEP
  LOOP WHILE RRATE < 0 OR RRATE > 255
  CALL SetRamp(SF3, SF2, SF1, RRATE)

CASE 14 'Get Digital Input State
  CALL ClearText
  CALL MoveCursor
  IF InitFlag = 1 THEN
    DO
      INPUT "Enter digital input number (1-11) "; DINBR
      IF DINBR < 1 OR DINBR > 11 THEN BEEP
    LOOP WHILE DINBR < 1 OR DINBR > 11
    CALL GetInput(DINBR, InputValue)
    LOCATE 16, 5: PRINT "INPUT "; DINBR; " STATE = "; InputValue;
  ELSE
    LOCATE 16, 5
    PRINT "Command Aborted, Not Initialized!";
  END IF

CASE 15 'Change Digital Output
  CALL ClearText
  CALL MoveCursor
  IF InitFlag = 1 THEN
    DO
      INPUT "Enter digital output number (1-7) "; DONBR
      IF DONBR < 1 OR DONBR > 7 THEN BEEP
    LOOP WHILE DONBR < 1 OR DONBR > 7
    CALL MoveCursor
    INPUT "Enter state (0 or 1) "; STATE
    IF STATE < 0 OR STATE > 1 THEN STATE = 0
    CALL ChangeOutput(DONBR, STATE)
  ELSE
    LOCATE 16, 5
    PRINT "Command Aborted, Not Initialized!";
  END IF

CASE 16 'Change Auto drive(s)
  CALL ClearText
  CALL MoveCursor
  INPUT "Enable/Disable Auto Drive for Stepper 1 E/D "; A$

```

```

        IF A$ = "E" OR A$ = "e" THEN Auto1 = 1 ELSE Auto1 = 0
        CALL MoveCursor
        INPUT "Enable/Disable Auto Drive for Stepper 2 E/D "; A$
        IF A$ = "E" OR A$ = "e" THEN Auto2 = 1 ELSE Auto2 = 0
        CALL MoveCursor
        INPUT "Enable/Disable Auto Drive for Stepper 3 E/D "; A$
        IF A$ = "E" OR A$ = "e" THEN Auto3 = 1 ELSE Auto3 = 0
        CALL ChangeAuto(Auto3, Auto2, Auto1)

    END SELECT

LOOP

END

SUB ChangeAuto (Auto3, Auto2, Auto1)

    Command = 14 'Change Auto Drive command number
    PRINT #1, CHR$((Auto3 * 64) + (Auto2 * 32) + (Auto1 * 16) + Command);

END SUB

SUB ChangeOutput (DONBR, STATE)

    Command = 13: PRINT #1, CHR$(DONBR * 16 + Command);
    PRINT #1, CHR$(STATE);

END SUB

SUB ClearText

    LOCATE 16, 5: PRINT STRING$(50, " ")

END SUB

SUB GetInput (DINBR, InputValue)

    Command = 12: PRINT #1, CHR$(DINBR * 16 + Command);
    InputValue = ASC(INPUT$(1, #1)) ' get input state

END SUB

SUB GetStatus (SF3, SF2, SF1, ReceivedStatus)

    Command = 10 'Get Stepper Status command number
    PRINT #1, CHR$((SF3 * 64) + (SF2 * 32) + (SF1 * 16) + Command);
    ReceivedStatus = ASC(INPUT$(1, #1)) ' get stepper status

END SUB

SUB GetStepsRemaining (SF3, SF2, SF1, StepsRemaining&)

    Command = 138 'Get Steps Remaining, command 10 with bit 7 set
    PRINT #1, CHR$((SF3 * 64) + (SF2 * 32) + (SF1 * 16) + Command);

```

```

    LoByte = ASC(INPUT$(1, #1)) ' Get the low step byte.
    HiByte = ASC(INPUT$(1, #1)) ' Get the high step byte.
    StepsRemaining& = (HiByte * 256) + LoByte

END SUB

SUB InitializeLimits (DO811, Limits3, Limits2, Limits1)

    Command = 2 'Initialize Limits command number
    PRINT #1, CHR$((DO811 * 128) + (Limits3 * 64) + (Limits2 * 32) + (Limits1 * 16) +
Command);

END SUB

SUB InitializeMode (HalfStep3, HalfStep2, HalfStep1)

    Command = 1 'Initialize Half Step command number
    PRINT #1, CHR$((HalfStep3 * 64) + (HalfStep2 * 32) + (HalfStep1 * 16) + Command);

END SUB

SUB InitializePhases (Phases3, Phases2, Phases1)

    Command = 3 'Initialize Phases command number
    PRINT #1, CHR$((Phases3 * 64) + (Phases2 * 32) + (Phases1 * 16) + Command);

END SUB

SUB MoveCursor

    ' Moves Cursor to command prompt line
    LOCATE 13, 32
    PRINT STRING$(42, " ")
    LOCATE 13, 20

END SUB

SUB ResetInterface (ResetResponse)

    PRINT #1, CHR$(80); ' send first byte of reset.
    PRINT #1, CHR$(95); ' send second byte of reset.

    ResetResponse = ASC(INPUT$(1, #1)) ' get reset response.

END SUB

SUB RunStepper (SF3, SF2, SF1)

    Command = 136 'Run Stepper command, Command 8 with bit 7 set
    PRINT #1, CHR$((SF3 * 64) + (SF2 * 32) + (SF1 * 16) + Command);

END SUB

```

```

SUB SetDirection (Dir3, Dir2, Dir1)

    Command = 7 'Change Direction command number
    PRINT #1, CHR$((Dir3 * 64) + (Dir2 * 32) + (Dir1 * 16) + Command);

END SUB

SUB SetDivisor (SF3, SF2, SF1, SPEED)

    Command = 5 'Change Speed Divisor command number
    PRINT #1, CHR$((SF3 * 64) + (SF2 * 32) + (SF1 * 16) + Command);
    PRINT #1, CHR$(SPEED);

END SUB

SUB SetRamp (SF3, SF2, SF1, RRATE)

    Command = 11 'Change Ramp Rate command number
    PRINT #1, CHR$((SF3 * 64) + (SF2 * 32) + (SF1 * 16) + Command);
    PRINT #1, CHR$(RRATE);

END SUB

SUB SetStepCount (SF3, SF2, SF1, Steps&)

    Command = 6 'Load Step Count command number
    PRINT #1, CHR$((SF3 * 64) + (SF2 * 32) + (SF1 * 16) + Command);
    ADDR = VARPTR(Steps&)
    LOSTEPS = PEEK(ADDR)
    PRINT #1, CHR$(LOSTEPS);
    HISTEPS = PEEK(ADDR + 1)
    PRINT #1, CHR$(HISTEPS);

END SUB

SUB SetStepRate (rate)

    Command = 4 'Change Step Rate Clock command number
    PRINT #1, CHR$(Command);
    PRINT #1, CHR$(rate);

END SUB

SUB StartStepper (SF3, SF2, SF1)

    Command = 8 'Start Stepper command number
    PRINT #1, CHR$((SF3 * 64) + (SF2 * 32) + (SF1 * 16) + Command);

END SUB

SUB StopStepper (SF3, SF2, SF1)

    Command = 9 'Stop Stepper command number
    PRINT #1, CHR$((SF3 * 64) + (SF2 * 32) + (SF1 * 16) + Command);

END SUB

```

Tips for salvaged stepper motors

The typical 4-phase unipolar stepper motor has six wires. These stepper motors have two separate center tapped coils. If you use this simple procedure you can get any 6-wire 4-phase stepper motor rotating in just a few minutes even if you do not have a wiring diagram for the motor. With your Volt/Ohm meter on a low ohms setting, find the two separate coils and their center tapped wires. Connect the wires at the center tap of each coil to the stepper motor power supply positive output V+ terminal on the SSC3 interface pc board. The two remaining wires of one coil are stepper phases 1 and 3. The two remaining wires of the other coil are stepper phases 2 and 4. Pick either one of the coils and hook the two wires up to the P1 and P3 output terminals of the pc board. Take the two wires from the remaining coil and connect them to the P2 and P4 terminals. At this point the motor should rotate, so give it a try! If the direction of rotation is the opposite that it is suppose to be then reverse the wires on terminals P1 and P3, or reverse the wires on terminals P2 and P4, but not both. Label your wires! If this procedure does not produce a smoothly rotating motor then you may have problems with the interface circuitry, the power supply, or perhaps you are trying to step the motor too fast. This procedure will get your stepper motor stepping even though the wires attached to terminals P1 through P4 may not necessarily be what the manufacturer states is the proper phase connections. The motor should step properly in either case. What is necessary is that the phases are pulsed in an even and odd phase sequence such as P3, P2, P1, P4 or P2, P3, P4, P1. In 2-phase excitation mode there is always one even and one odd phase energized at the same time.

Some four phase stepper motors have five wires. The five wire motors have their center taps joined together inside the motor. You will not be able to find the even and odd coil pairs of these motors using a volt/ohm meter. You will only be able to find the wire that is common to all phases. However, if both of the even or both of the odd phases are energized at the same time, the motor will have no holding torque. There will always be holding torque when any one phase is energized, or when 1 even and 1 odd phase are energized together.

The program SSCPROG.BAS provided on the programming examples disk will generate a continuous step pattern for you while you are trying to determine the stepper motors proper connections.