

# SS-SC11 Manual

A Super Stepper Command Bus Architecture Product



Tune to [www.SuperStepper.com](http://www.SuperStepper.com) for quick tutorials on how to get the best of the Super Stepper Architecture!



# 11 RC Servo Controller - Super Stepper Architecture



**Super Stepper RC Servo Controller (SS-SC11):**  
Control 11 RC Servos position individually.  
Enable and disable any servo individually.

The SS-SC11 RC Servo controller allows the user to control up to 11 RC Servo motors individually. The user can enable and disable any combination of servos as well as control the desired position with 8 bit accuracy offering up to 256 different positions.

Since servos have the tendency to move to the commanded position as soon as the command is presented as an input, the user may want to chop the command into little segments and control the rate of change. This controller offers an automated way of controlling the rate on an individual servo basis. Talk about flexibility!

The board works with 12V. An extra high current connector is supplied to allow the user to supply RC Servo motor power (usually from 4V to 6V).

The module receives command through the SSB (Super Stepper Bus). Two jumpers, JMP1 and JMP2, configure the controller UART for either 8 or 9 bit communication and a different range of BAUD Rates and addresses.

Commands received through the bus are executed immediately. Available commands for the SS-SC11 are shown on Table 1.

Porta Super Servo 11							
Opcode	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Description	# of Bytes
0	EN/DIS	-	-	-	-	Enable/Disable Servo	1
1	Servo #	Position	-	-	-	Move Servo	2
2	-	-	-	-	-	Reserved	1
3	-	-	-	-	-	Reserved	1
4	-	-	-	-	-	Reserved	1
5	Expanded	-	-	-	-	Reserved	1
6	Mem Sel	Address	Data	-	-	Write	4
7	Mem Sel	Address	-	-	-	Read	3

Table 1

# 11 RC Servo Controller - Super Stepper Architecture

---

## Commands:

**Enable / Disable Servo** (Opcode 0): This command tells the controller to enable or disable all servos. The En/Dis\* bit in the parameter byte determines desired function (0 = disable, 1 = enable). The disable function works on all servos at once. The enable function works on all servos that have been masked to be enabled on the Enabled Servo Mask word. Any servo with a 1 in its respective Enabled Servo Mask bit, will be enabled.

Once the servos are enabled, they moved to their programmed position. The positions can be programmed before the servos are enabled. Figure 1 shows the command structure. Table 2 shows En/Dis\* bit description.

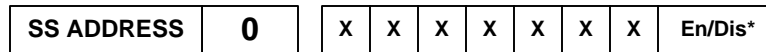


Figure 1

En/Dis*	Action
0	Disable All Servos
1	Enable All Servos

Table 2

## Enabled Servo Mask:

Each servo can be individually enabled thanks to the Enabled Servo Mask found on EEPROM and RAM. These word contains 11 bits, one per servo. Each respective servo will receive its PWM signal if the bit is set (SRVx= 1). These bits can be modified at run time by writing to RAM.

The EEPROM contains two bytes denominated Enabled Servo Mask Hi and Lo. These two bytes are factory preset to 0x07FF. After power up, the bytes are transferred to RAM where the controller uses the information to determine enabled feature. It is recommended that the user burns the desired setting on the corresponding EEPROM space, whenever it is to change. Or, soon after powering up the application, the RAM must be initialized with the correct value.

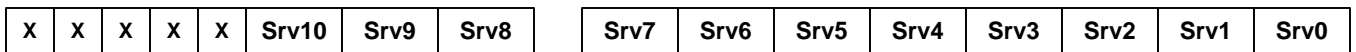


Figure 2

Refer to the EEPROM and RAM Memory Map sections for more information on these important variables, including respective addresses.

# 11 RC Servo Controller - Super Stepper Architecture

**Move Servo X** (Opcode 1): This command modifies the internal servo position register on an individual servo basis. If the particular servo is enabled, the new commanded position will become apparent right away. On the other hand, if the servo is disabled, the commanded speed will be stored in memory until it is changed again. The servo will then move once it is enabled.

The parameters sent are as follow:

**Servo #:** Specifies a servo from 0 to 10. Sending numbers larger than 10 may result in undesirable operation.

**Servo Position:** A number from 0 to 255 which specifies the desired position. 0 corresponds to full left, while 255 corresponds to full right.



Figure 3

**Write RAM/EEPROM:** (Opcode 6) Writes to internal RAM or EEPROM memory. The MEM SEL parameter specifies which type of memory will be accessed. Logic Low (0) accesses RAM while Logic High (1) accesses EEPROM. All writes are done to byte memory spaces. Writing to a word is not supported, so user must do two consecutive word writes to write to a word memory allocation, if needed.

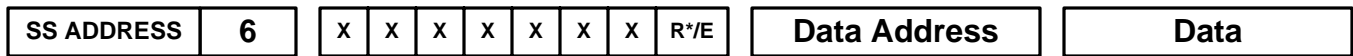


Figure 4

**Read RAM/EEPROM:** (Opcode 7) Reads from internal RAM or EEPROM memory. The MEM SEL parameter specifies which type of memory will be read and if the read request will return a word or a byte. Only reads to RAM can return a word. EEPROM reads always returns a byte.

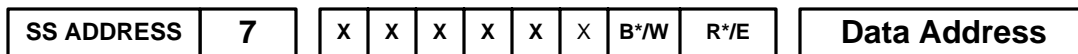


Figure 5

B*/W	R*/E	Description
0	0	Read a RAM Byte from ADDRESS
0	1	Read an EEPROM Byte from ADDRESS
1	0	Read a RAM Word (16bit) from ADDRESS
1	1	Read an EEPROM Byte from ADDRESS

Table 3

# 11 RC Servo Controller - Super Stepper Architecture

---

## RC Servo Primer:

RC Servos are considered DC Brush motor servos because they work under the principle of accurately obtaining a desired position with feedback. Here is how it works! The RC Servo receives a position command encoded in a pulse. The signal is called Pulse Width Modulated (PWM) because by changing the pulse width, a different command is specified. Refer to the following figure for a sketch of a PWM signal.

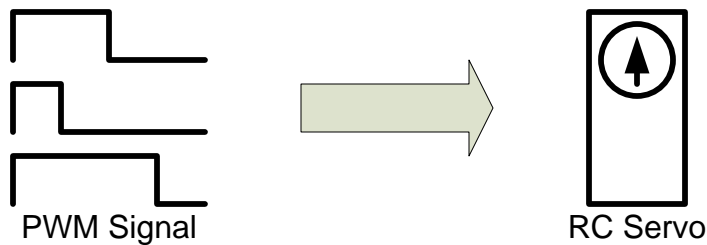


Figure 6

Inside the RC Servo there is circuitry to interpret the pulse width. At the same time, the motor shaft is tied to a potentiometer (variable resistor). The potentiometer offers an output voltage to the control circuitry which is proportional to the current shaft position. The internal control circuit compares the current position voltage to the pulse width. If the pulse width contains a different command position to the actual position specified by the potentiometer voltage, the controller starts to move the motor shaft in the direction in which the command position is to be found.

This is known as a closed loop system because the potentiometer voltage is a feedback utilized by the controller to try and attain a commanded position. The block diagram seen in the figure below depicts such idea.

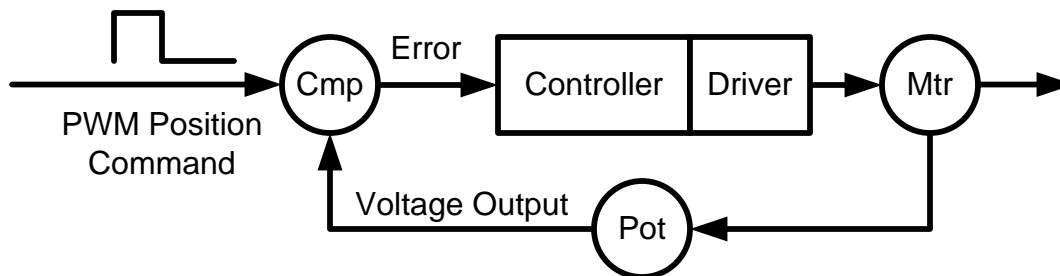


Figure 7

# 11 RC Servo Controller - Super Stepper Architecture

---

## RC Servo Primer:

The error is what the controller tries to minimize in order to fully utilize the feedback information and move the shaft to the desired position. Hence, if the desired position is very far away from the actual position, the controller knows how to apply power in the same direction needed for the two positions to become the same and the error smaller. Once the error approaches zero, the controller diminishes the motor speed until the desired position and the actual position become exactly the same.

The controller is at all times monitoring the encoded desired position (PWM) and the actual position (Potentiometer). As soon as one of these values changes, the controller will try to correct and minimize the error. Per example, if the motor has acquired the correct position, but something tries to move the shaft, the motor will compensate in order for the shaft to remain or return to the desired position. Also, if the commanded position changes (i.e. the PWM changes) the controller will then again move the shaft until the goal is achieved.

# 11 RC Servo Controller - Super Stepper Architecture

---

## Understanding RC Servo PWM and Command Position:

The encoding method used by excellence with RC products is the 1.5ms pulse. That is, a 1.5 ms pulse is modified up to .5 ms up or down to signal different commands. In the case of the servo, the 1.5 ms pulse width specifies center position, 1 ms specifies full left position and 2 ms specifies full right position. The user must have in mind that this scheme applies for RC Servos but other RC products utilizes other variants such as proportional speed with direction control, the one found on RC car speed controllers and modified rotational RC Servos.

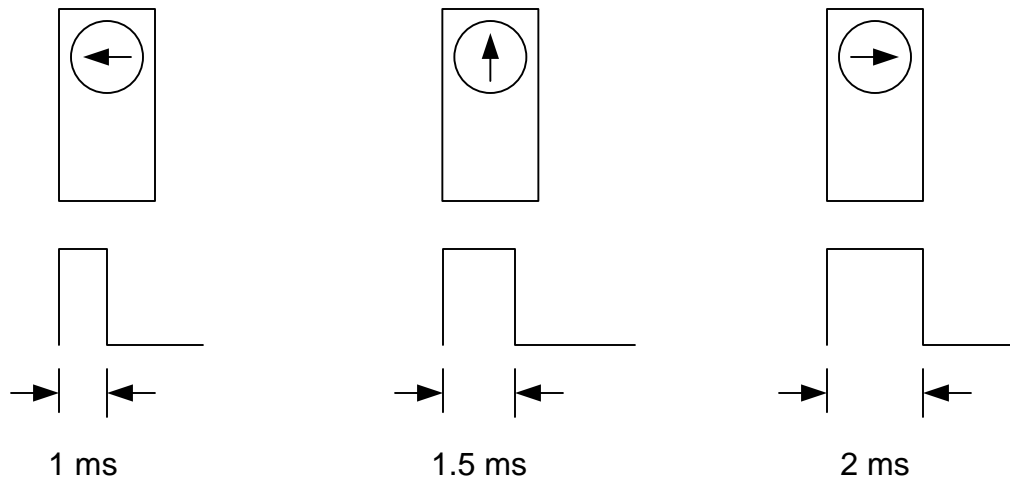


Figure 8

The figure above illustrates the PWM scheme. Notice that 1.0 ms corresponds to one side, while 2.0 ms corresponds to the opposite one. Which side the Servo moves with 1.0 ms and 2.0 ms commands, depends on the manufacturer.

The SS-SC11 can control up to 11 servos by generating the modulated 1.5 ms pulse width. Each servo has their own pulse being generated. Hence, each servo will pursue a totally independent commanded position. The pulse is generated every 22 ms in order to allow all servos to obtain full right position if necessary.

The SS-SC11 can generate 256 pulse widths as specified by the Position Byte. This gives enough resolution to control practically any RC servo application.

# 11 RC Servo Controller - Super Stepper Architecture

---

## Servo Rate of Change

Servos attempt to move to the command position as soon as the PWM signal changes to the command position. This implies that as soon as the PWM changes to 1.5 ms (with command = 128), per example, the servo motor will try to move to the center position right away. In all systems this clearly generates a jerky motion. Delicate systems may not tolerate such abrupt change of position.

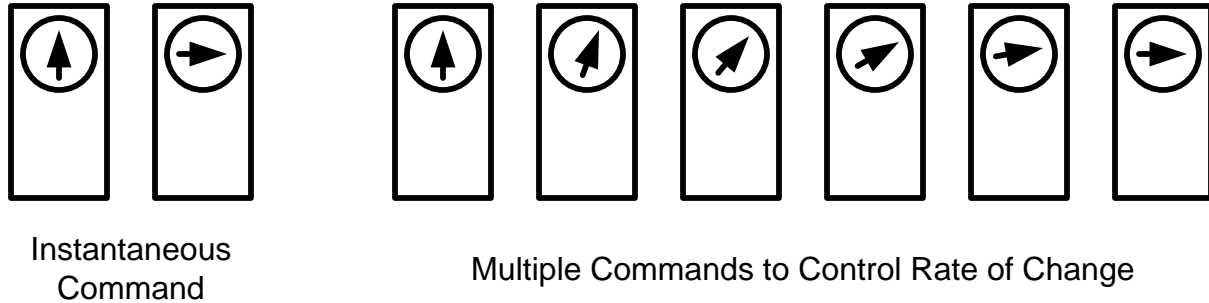


Figure 9

As can be seen on the picture above, a solution to this problem is to command the motor to move in smaller segments through time. In other words if the current position is 100 and the desired position is 128, the user may command 101, 102 and so on until reaching 128. In this previous example it is said that the rate of change is 1 unit. The time base for the rate of change is how fast each command is sent to the controller.

Such technique represents an undesirable scenario due to timing constraints and number of messages per motor to control rate of change. Consider that if you the user is to control the rate of change of the 11 servos, the serial bus will be crowded with messages per position command.

Instead, the SS-PortaSC11 contains a feature to control each servo motor's rate of change. Rate of Change Time Base Register (address 0x02 in RAM) specifies how many milliseconds will elapse from one position command to the next. The Time Base is one for all servo motors. Placing a zero in this register is considered to be the same as 1 millisecond. Any other number equals itself in milliseconds.

Each RC Servo channel has its own Rate of Change Register (0x03 to 0x0D). This implies that some servos can be updated faster than others. Writing a zero to the Rate of Change Register is taken as a 1. Every other number represents itself in position command units. Per example if Servo0's Rate of Change Register has a 10, its position will be updated by 10 units until reaching the desired position.

# 11 RC Servo Controller - Super Stepper Architecture

---

## Servo Rate of Change Registers

ROM Addr	RAM Addr	Contents	Value
0x04	0x02	Rate of Change Time Base Register	0 to 255
0x05	0x03	Servo 0 Rate of Change Register	0 to 255
0x06	0x04	Servo 1 Rate of Change Register	0 to 255
0x07	0x05	Servo 2 Rate of Change Register	0 to 255
0x08	0x06	Servo 3 Rate of Change Register	0 to 255
0x09	0x07	Servo 4 Rate of Change Register	0 to 255
0x0A	0x08	Servo 5 Rate of Change Register	0 to 255
0x0B	0x09	Servo 6 Rate of Change Register	0 to 255
0x0C	0x0A	Servo 7 Rate of Change Register	0 to 255
0x0D	0x0B	Servo 8 Rate of Change Register	0 to 255
0x0E	0x0C	Servo 9 Rate of Change Register	0 to 255
0x0F	0x0D	Servo 10 Rate of Change Register	0 to 255

Table 4

The table above shows all the register needed to configure the Rate Of Change engine. After power up, the ROM contents are transferred to the respective RAM location for automatic initialization. The user may change the RAM location for immediate update. ROM values changed will not be available to the user until the next power cycle up.

# 11 RC Servo Controller - Super Stepper Architecture

---

## BAUD Rate Selection:

The SS-SC11 board communicates through the Super Stepper Serial Communication Protocol. Since the board has been implemented with a 4MHz crystal, certain BAUD Rates can be obtained. Programming the EEPROM with one of the values found on Table 3 will configure the UART to operate with the selected BAUD Rate after the device comes out of power on reset next time. This feature is of course available if the JMP1 is closed.

Baud Rate	EEPROM
2400	103
4800	51
9600	25
14400	16
19200	12
28800	8
38400	6
57600	3
76800	2
115200	1

Table 5

## EEPROM Memory Map:

The user must refer to this section whenever the EEPROM is to be programmed. Programming the EEPROM memory on the device may result on non proper functionality.

EEPROM Address	Contents	EEPROM Address	Contents
0x00	SS BAUD Rate	0x04	Rate Change Time Base
0x01	SS Address	0x05 to 0x0F	Rate Change Magnitude
0x02	Servo Enable Mask Hi	0x10 to 0xFF	User defined
0x03	Servo Enable Mask Lo		

Table 6

## RAM Memory Map:

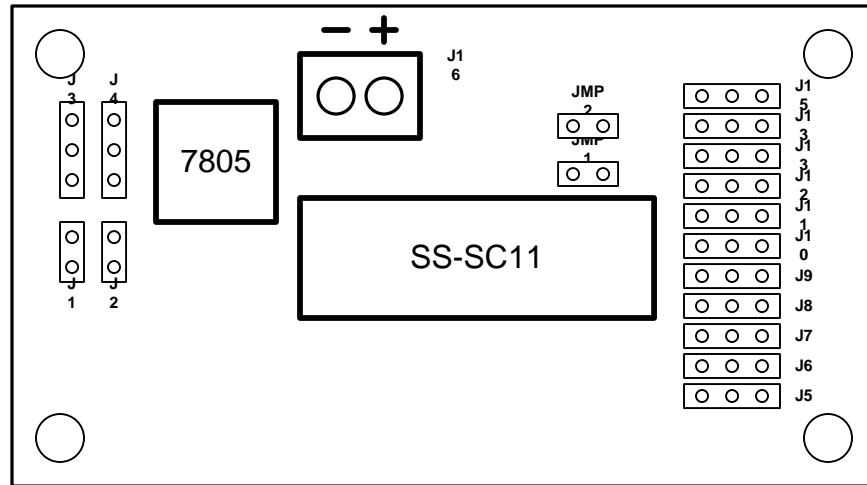
The user must refer to this section whenever the RAM is to be read or written. Modifying RAM memory on the device could result on non proper functionality.

RAM Address	Contents	RAM Address	Contents
0x00	Servo Enable Mask Hi	0x03 to 0x0D	Rate Change Magnitude
0x01	Servo Enable Mask Lo	0x0E to 0x17	User defined
0x02	Rate Change Time Base	0x18 to 0x1F	Do not Use

Table 7

# 11 RC Servo Controller - Super Stepper Architecture

---



J1 and J2 Power Connector. Connect a power source from 9V to 15V. J2 is a bypass to another Porta Board piggy backed on top.

- Pin1: Positive Power Source
- Pin2: Ground

J3 and J4 Serial Communications Connector: Connect the TTL level serial communication signal as obtained from a PC through an RS-232 driver or any microcontroller UART output.

- Pin1: Module Receive (Rx)
- Pin2: Module Transmit (Tx)
- Pin3: Ground

J5 through J15 RC Servo PWM output. Connect the three pin connector found on RC servos. Watch polarity as shown below. Notice Servo 0 is the connector found on the lowest section while Servo 10 is the upper most.

- Pin1: Servo Ground (as marked with a - sign)
- Pin2: Servo Power (as marked with a + sign)
- Pin3: Servo PWM Output (as marked with a pulse symbol)

J16 Servo Power Connector: Connect 4V to 6V power source to be routed into the 11 RC servo bank. Make sure the Servo power supply is well regulated and can supply large currents as enabled servos sink around 250 mA. The connector is good up to 9 Amps. This connector is not polarity protected. Failure to connect it as specified will result in C8 being damaged.

JP1 MOD1 Jumper. Selects from UART hardwired parameters or EEPROM stored parameters.

- Open: BAUD Rate is 2400 and SS-ADDRESS is 0.
- Closed: BAUD RATE is stored on EEPROM address 0 and SS-ADDRESS is stored on EEPROM address 1.

JP2 MOD0 Jumper. Selects from 8 bit or 9 bit communications.

- Open: 8 bit communications selected.
- Closed: 9 bit communications selected.