SSC-32 Manual.

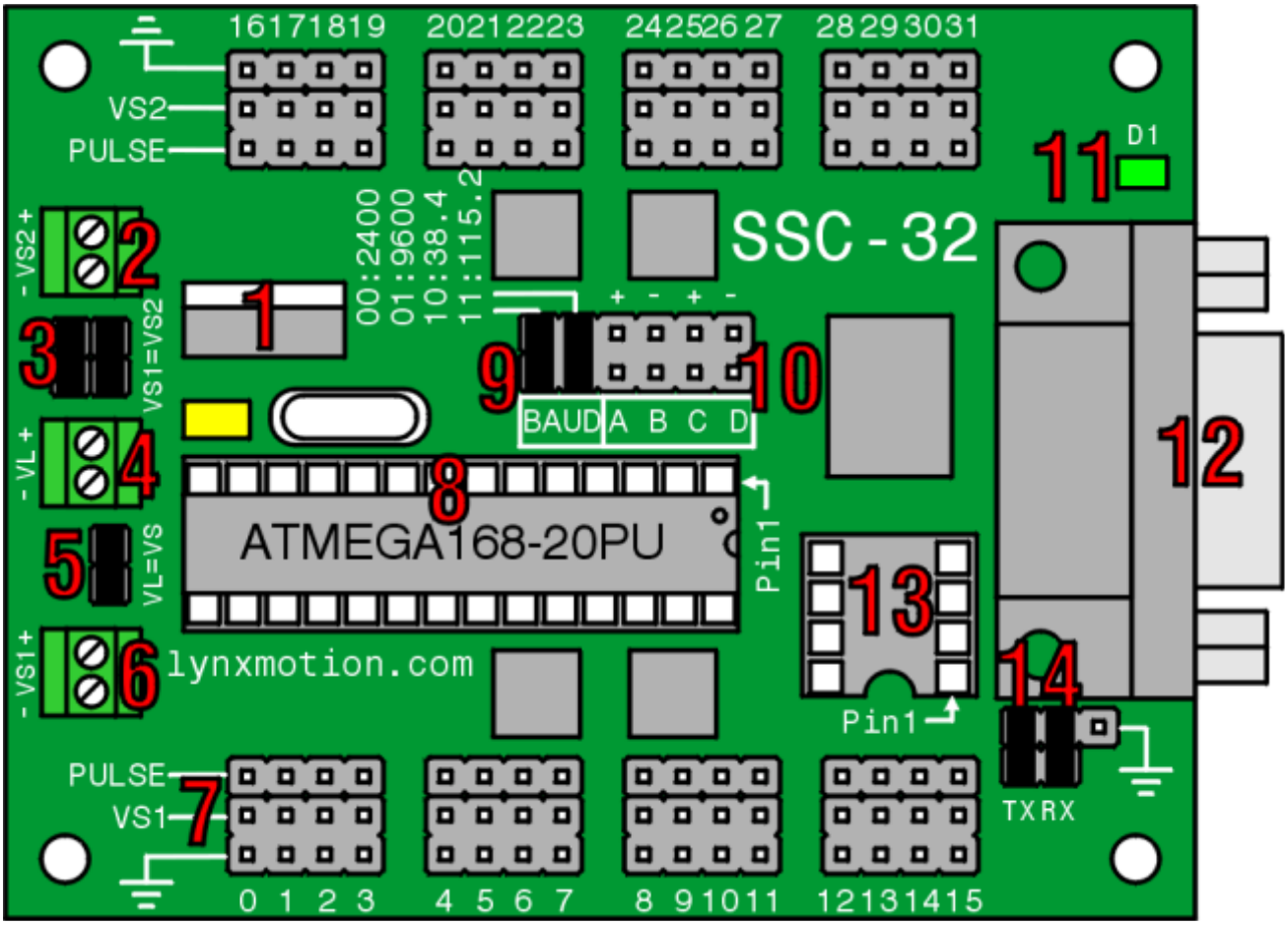
Author: Jim Frye
Version: 2.01XE
Date: June 16, 2010

Table of Contents

Links

- SSC-32 Schematic (pdf)
- SSC-32 (v2) GP Sequencer Usage Manual
- LynxTerm Download

SSC-32.



SSC-32 Hardware Information.

The image above illustrates the shorting bar jumpers (the black rectangles) as they are on the board as it is shipped. The jumpers are used to set operating parameters for the board. It should not be assumed that they are correct for your project as changes are likely to be required. Please consult the tutorial for your project for the proper shorting bar jumper positions.

The SSC-32 is 3.00" x 2.30" with 0.125" holes set in 0.15" from each edge.

1. The Low Dropout regulator will provide 5vdc out with as little as 5.5vdc coming in. This is important when operating your robot from a battery. It can accept a maximum of 9vdc in. The regulator is rated for 500mA, but we are de-rating it to 250mA to prevent the regulator from getting too hot.
2. This terminal connects power to servo channels 16 through 31. Apply 4.8vdc to 6.0vdc for most analog or digital servos. This can be directly from a 5-cell NiMH battery pack. 7.2vdc - 7.4vdc can be applied to HSR-5980 or HSR-5990 servos. This can be directly from a 6-cell NiMH battery pack or a 2-cell LiPo battery pack.

Board	Input
VS2 +	RED
VS2 -	BLACK

3. These jumpers are used to connect VS1 to VS2. Use this option when you are powering all servos from the same battery. Use both jumpers. Alternately, if you want to use two separate battery packs, one on each side, then remove both of these jumpers.
4. This is the Logic Voltage, or VL. This input is normally used with a 9vdc battery connector to provide power to the ICs and anything connected to the 5vdc lines on the board. The valid range for this terminal is 6vdc - 9vdc. This input is used to isolate the logic from the Servo Power Input. It is necessary to remove the VS1=VL jumper when powering the servos separately from the logic VL. The SSC-32 should draw 35mA with nothing connected to the 5vdc output.

Board	Input

VL +	RED
VL -	BLACK

5. This jumper allows powering the microcontroller and support circuitry from the servo power supply. This requires at least 6vdc to operate correctly. If the microcontroller resets when too many servos are moving at the same time, it may be necessary to power the microcontroller separately using the VL input. A 9vdc works nicely for this. This jumper must be removed when powering the microcontroller separately!
6. This terminal connects power to servo channels 16 through 31. Apply 4.8vdc to 6.0vdc for most analog or digital servos. This can be directly from a 5-cell NiMH battery pack. 7.2vdc - 7.4vdc can be applied to HSR-5980 or HSR-5990 servos. This can be directly from a 6-cell NiMH battery pack or a 2-cell LiPo battery pack.

Board	Input
VS1 +	RED
VS1 -	BLACK

7. This is where you connect the servos or other output devices. Use caution and remove power when connecting anything to the I/O bus.

   For discrete outputs (Hi / Low), each output can sync or source up to 35mA. However there is a maximum limit of 70mA per 8 I/O pin group. I.e. 0-7, 8-15, 16-23, 24-31.

Board	Wire
Pulse	Yellow or White
VS	Red
Ground	Black or Brown

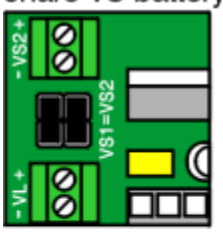
8. This is where the Atmel IC chip goes. Be careful to insert it with Pin 1 in the upper right corner as pictured. Take care to not bend the pins.
9. The two BAUD inputs allow configuring the baud rate. Please see the examples below.

Jumpers	Baud Rate	Usage
0 0	2400	Slower Processors
0 1	9600	Slower Processors
1 0	38.4k	Atom/Stamp Communication
1 1	115.2k	PC Communication, Firmware Update

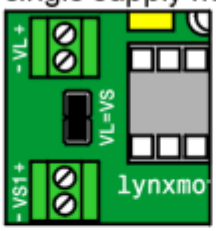
10. The ABCD inputs have both static and latching support. The inputs have internal weak (50k) pullups that are used when a Read Digital Input command is used. A normally open switch connected from the input to ground will work fine.
11. This is the Processor Good LED. It will light steady when power is applied and will remain lit until the processor has received a valid serial command. It will then go out and will blink whenever it is receiving serial data.
12. Simply plug a straight-through M/F DB9 cable from this plug to a free 9-pin serial port con your PC for receiving servo positioning data. Alternately a USB-to-serial adapter will work well. Please see the USB-to-serial section for more information.
13. This is an 8-pin EEPROM socket. The EEPROM is supported by the 2.01GP firmware.
14. This is the TTL serial port or DB9 serial port enable. Install two jumpers as illustrated below to enable the DB9 port. Install wire connectors to utilize TTL serial communication from a host microcontroller.

Shorting Bar Jumpers and Connectors at a glance.

Applies VS1 to VS2 share VS battery.

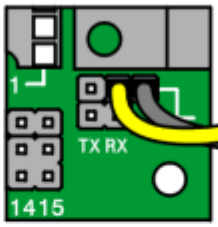


Applies VS to VL single supply mode.



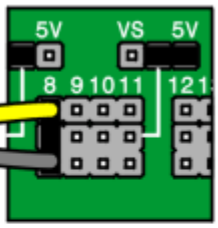
Unidirectional TTL Serial Communications.

SSC-32 side...

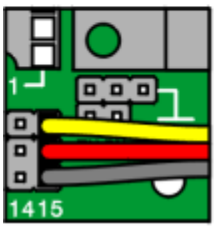


DC-01

Bot Board side...

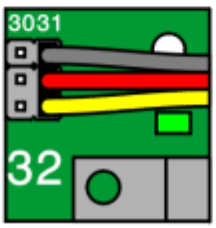


Example servo connection 0-15.



Yellow
Red
Black

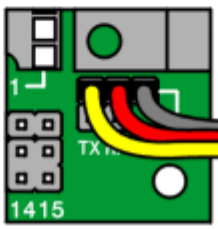
Example servo connection 16-31.



Black
Red
Yellow

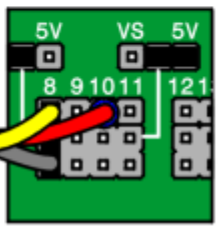
Bidirectional TTL Serial Communication.

SSC-32 side...

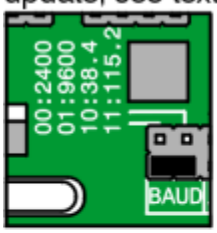


Modified SC-01

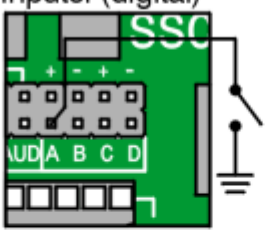
Bot Board side...



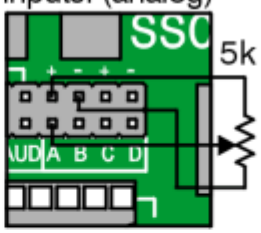
Force firmware update, see text!



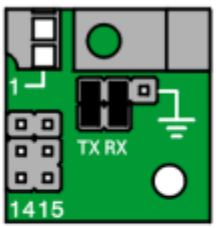
ABCD auxiliary inputs. (digital)



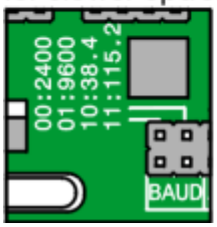
ABCD auxiliary inputs. (analog)



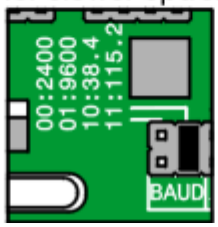
DB9 enable for PC use.



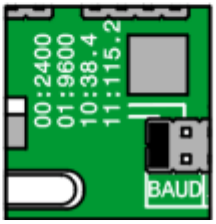
Baud rate 2400 for slower cpu's.



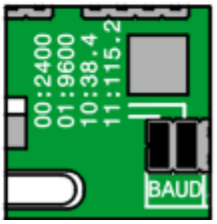
Baud rate 9600 for slower cpu's.



Baud rate 38.4k for Basic Atom use.



Baud rate 115.2k for PC use.



Command Formatting for the SSC-32.

Command Types and Groups.

With the exception of MiniSSC-II mode, all SSC-32 commands must end with a carriage return character (ASCII 13). Multiple commands of the same type can be issued simultaneously in a *Command Group*. All of the commands in a command group will be executed after the final carriage return is received. Commands of different types cannot be mixed in the same command group. In addition, numeric arguments to all SSC-32 commands must be ASCII strings of decimal numbers, e.g. "1234". Some commands accept negative numbers, e.g. "-5678". Programming examples will be provided. ASCII format is not case sensitive. Use as many bytes as required. Spaces, tabs, and line feeds are ignored.

Command Types and Groups			
1	Servo Movement	7	Read Analog Inputs
2	Discrete Output	8	12 Servo Hexapod Gait Sequencer
3	Byte Output	9	Query Hex Sequencer
4	Query Movement Status	10	Get Version
5	Query Pulse Width	11	Go to Boot

6	Read Digital Inputs	12	MiniSSC-II Compatibility

Servo Move or Group Move.

# <ch> P <pw> S <spd> ... # <ch> P <pw> S <spd> T <time> <cr>	
<ch>	Channel number in decimal, 0-31
<pw>	Pulse width in microseconds, 500-2500
<spd>	Movement speed in uS per second for one channel (Optional)
<time>	Time in mS for the wntire move, affects all channels, 65535 max (Optional)
<cr>	Carriage return character, ASCII 13 (Required to initiate action)
<esc>	Cancel the current action, ASCII 27

Servo Move Example: "#5 P1600 S750 <cr>"

The example will move the servo on channel 5 to position 1600. It will move from its current position at a rate of 750uS per second until it reaches its commanded destination. For a better understanding of the speed argument, consider that 1000uS of travel will result in around 90□ of rotation. A speed value of 100uS per second means the servo will take 10 seconds to move 90□. Alternately, a speed value of 2000uS per second equates to 500mS (half a second) to move 90□.

Servo Move Example: "#5 P1600 T1000 <cr>"

The example will move servo 5 to position 1600. It will take 1 second to complete the move regardless of how far the servo has to travel to reach the destination.

Servo Group Move Example: "#5 P1600 #10 P750 T2500 <cr>"

The example will move servo 5 to position 1600 and servo 10 to position 750. It will take 2.5 seconds to complete the move, even if one servo has farther to travel than another. The servos will both start and stop moving at the same time. This is a very powerful command. By commanding all the legs in a walking robot with the Group Move it is easy to synchronize complex gaits. The same synchronized motion can benefit the control of a robotic arm as well.

You can combine the speed and time commands if desired. The speed for each servo will be calculated according to the following rules:

1. All channels will start and end the move simultaneously.
2. If a speed is specified for a servo, it will not move any faster than the speed specified (but it might move slower if the time command requires).
3. If a time is specified for the move, then the move will take at least the amount of time specified (but might take longer if the speed command requires).

Servo Move Example: "#5 P1600 #17 P750 S500 #2 P2250 T2000 <cr>"

The example provides 1600uS on ch5, 750uS on ch17, and 2250uS on ch2. The entire move will take at least 2 seconds, but ch17 will not move faster than 500uS per second. The actual time for the move will depend on the initial pulse width for ch17. Suppose ch17 starts at position 2000. Then it has to move 1250uS. Since it is limited to 500uS per second, it will require at least 2.5 seconds, so the entire move will take 2.5 seconds. On the other hand, if ch17 starts at position 1000, it only needs to move 250uS, which it can do in 0.5 seconds, so the entire move will take 2 seconds.

Important! The first positioning command should be a normal "# <ch> P <pw>" command. Because the controller doesn't know where the servo is positioned on powerup, it will ignore speed and time commands until the first normal command has been received.

Any move that involves more than one servo and uses either the S or T modifier is considered a Group Move, and all servos will start and stop moving at the same time. If you require moving several servos at different speeds, you must issue the commands separately.

Software Position Offset.

# <ch> PO <offset value> ... # <ch> PO <offset value> <cr>	

<ch>	Channel number in decimal, 0-31
<offset value>	100 to -100 in uSeconds
<cr>	Carriage return character, ASCII 13

This command allows the servos centered (1500uS) position to be aligned perfectly. The servo channel will be offset by the amount indicated in offset value. This represents approximately 15□ of range. It's important to build the mechanical assembly as close as possible to the desired centered position before applying the servo offset. This makes it easy to setup servos that do not have mechanical alignment. The Position Offset command should be issued only once in your program. When the SSC-32 is turned off it will forget the Position Offsets.

The current SSC-32 now has an internal register method for doing Position Offsets. These are stored in the Atmel chips internal EEPROM and are retained when power is removed. Use of this feature is covered in the Register Support section of this manual.

General Output Information.

The outputs on the SSC-32 come from four 74HC595 8 bit shift register chips. There are four banks of 8 bit outputs as shown 0-7, 8-15, 16-23 and 24-32. The outputs can sink or source up to 25mA per pin, but a max of 70mA per bank must be observed.

Discrete Output.

# <ch> <lvl> ... # <ch> <lvl> <cr>	
<ch>	Channel number in decimal, 0-31
<lvl>	Logic level for the channel, either 'H' for High or 'L' for Low
<cr>	Carriage return character, ASCII 13

The channel will go to the level indicated within 20mS of receiving the carriage return.

Discrete Output Example: "#3H #4L <cr>"

This example will output a High (+5v) on channel 3 and a Low (0v) on channel 4.

Byte Output.

# <bank> : <value> <cr>	
<bank>	(0 = Pins 0-7, 1 = Pins 8-15, 2 = Pins 16-23, 3 = Pins 24-31)
<value>	Decimal value to output to the selected bank (0-255), Bit 0 = LSB of bank
<cr>	Carriage return character, ASCII 13

This command allows 8 bits of binary data to be written at once. All pins of the bank are updated simultaneously. The banks will be updated within 20mS of receiving the carriage return.

Bank Output Example: "#3:123 <cr>"

This example will output the value 123 (decimal) to bank 3. 123 (dec) = 01111011 (bin), and bank 3 is pins 24-31. So this command will output a "0" to pins 26 and 31, and will output a "1" to all other pins.

Query Movement Status.

Example: "Q <cr>"

This will return a "." if the previous move is complete, or a "+" if it is still in progress.

There will be a delay of 50uS to 5mS before the response is sent.

Query Pulse Width.

Example: "QP <arg> <cr>"

This will return a single byte (in binary format) indicating the pulse width of the selected servo with a resolution of 10uS. For example, if the pulse width is 1500uS, the returned byte would be 150 (binary).

Multiple servos may be queried in the same command. The return value will be one byte per servo. There will be a delay of at least 50uS to 5mS before the response is sent Typically the response will be started within 100uS.

Read Digital Inputs.

Example: "A B C D AL BL CL DL <cr> "

A, B, C, and D are normal input reads. They read the value on the input as a binary value. It returns ASCII "0" if the input is a low (0v) or an ASCII "1" if the input is a high (+5v).

AL, BL, CL, and DL are latching input reads. They return the value on the input as an ASCII "0" if the input is a low (0v) or if it has been low since the last *L command. It returns a high (+5v) if the input is high and never went low since the last *L command. Simply stated, it will return a low if the input ever goes low. Reading the status simply resets the latch.

The ABCD inputs have a weak pullup (~50k) that is enabled when used as inputs. They are checked approximately every 1mS, and are debounced for approximately 15mS. The logic value for the read commands will not be changed until the input has been at the new logic level continuously for 15mS. The Read Digital Input Commands can be grouped in a single read, up to 8 values per read. They will return a string with one character per input with no spaces.

Read Digital Input Example: "A B C DL <cr>"

Read Analog Inputs.

Example: "VA VB VC VD <cr>"

VA, VB, VC, and VD read the value on the input as analog. It returns a single byte with the 8-bit (binary) value for the voltage on the pin.

When the ABCD inputs are used as analog inputs the internal pullup is disabled. The inputs are digitally filtered to reduce the effect of noise. The filtered values will settle to their final values within 8mS of a change. A return value of 0 represents 0vdc. A return value of 255 represents +4.98vdc. To convert the return value to a voltage, multiply by 5/256. At power up the ABCD inputs are configured for digital input with pullup. The first time a V* command is used, the pin will be converted to analog without pullup. The result of this first read will not return valid data.

Read Analog Input Example: "VA VB <cr>"

This example will return 2 bytes with the analog values of A and B. For example is the voltage on Pin A is 2vdc and Pin B is 3.5vdc, the return value will be the bytes 102 (binary) and 179 (binary).

12 Servo Hexapod Sequencer Commands.

LH <arg>, LM <arg>, LL <arg>

Set the value for the vertical servos on the left side of the hexapod. LH sets the high value, i.e. the pulse width to raise the leg to its maximum height; LM sets the mid value; and LL sets the low value. The valid range for the arguments is 500 to 2500uS.

RH <arg>, RM <arg>, RL <arg>

Set the value for the vertical servos on the right side of the hexapod. RH sets the high value, i.e. the pulse width to raise the leg to its maximum height; RM sets the mid value; RL sets the low value. The valid range for the arguments is 500 to 2500uS.

VS <arg>

Sets the speed for movement of vertical servos. All vertical servo moves use this speed. Valid range is 0 to 65535uS/Sec.

LF <arg>, LR <arg>

Set the value for the horizontal servos on the left side of the robot. LF sets the front value, i.e. the pulse width to move the leg to the maximum forward position; LR sets the rear value. The valid range for the arguments is 500 to 2500uS.

RF <arg>, RR <arg>

Set the values for the horizontal servos on the right side of the robot. RF sets the front value, i.e. the pulse width to move the leg to the maximum forward position; RR sets the rear value. The valid range for the arguments is 500 to 2500uS.

HT <arg>

Sets the time to move between horizontal front and rear positions. The valid range for the argument is 1 to 65535uS.

XL <arg>, XR <arg>

Set the travel percentage for left and right legs. The valid range is -100% to 100%. Negative values cause the legs on the side to move in reverse. With a value of 100%, the legs will move between the front and rear positions. Lower values cause the travel to be proportionally less, but always centered. The speed for horizontal moves is adjusted based on the XL and XR commands, so the move time remains the same.

XS <arg>

Set the horizontal speed percentage for all legs. The valid range is 0% to 200%. With a value of 100%, the horizontal travel time will be the value programmed using the HT command. Higher values proportionally reduce the travel time; lower values increase it. A value of 0% will stop the robot in place. The hex sequencer will not be started until the XS command is received.

XSTOP

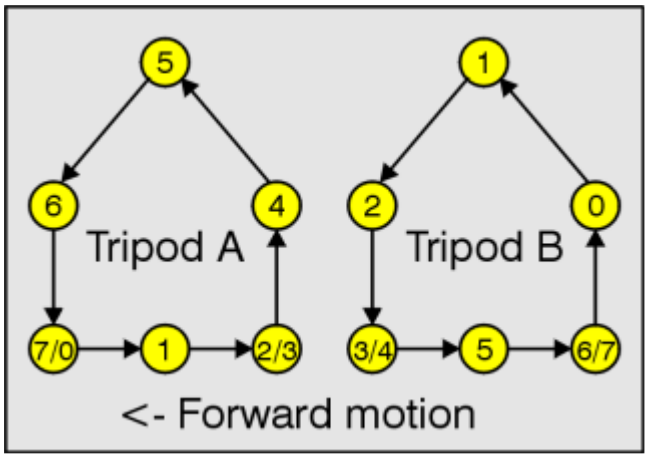
Stop the hex sequencer. Return all servos to normal operation.

Notes on Hexapod Sequencer.

1. The following servo channels are used for the hex sequencer:

0	Right Rear Vertical	16	Left Rear Vertical
1	Right Rear Horizontal	17	Left Rear Horizontal
2	Right Center Vertical	18	Left Center Vertical
3	Right Center Horizontal	19	Left Center Horizontal
4	Right Front Vertical	20	Left Front Vertical
5	Right Front Horizontal	21	Left Front Horizontal

2. The hexapod walking gait is an alternating tripod. The tripods are labeled Tripod A and Tripod B. Tripod A consists of {Left Front, Left Rear, Right Center}, and Tripod B consists of {Left Center, Right Front, Right Rear}.
3. While walking, the legs pass through 6 points: (Low Front), (Low Center), (Low Rear), (Mid Rear), (High Center), and (Mid Front). "Center" refers to the mid-point between the Front and Rear positions.



4. The walking sequence consists of 8 states, numbered 0-7. They are defined below:

State	Tripod A		Tripod B	
	Vertical	Horizontal	Vertical	Horizontal
0	Low	Front to Center	Mid to High	Rear to Center
1	Low	Center to Rear	High to Mid	Center to Front
2	Low	Rear	Mid to Low	Front
3	Low to Mid	Rear	Low	Front
4	Mid to High	Rear to Center	Low	Front to Center
5	High to Mid	Center to Front	Low	Center to Rear
6	Mid to Low	Front	Low	Rear
7	Low	Front	Low to Mid	Rear

   In this table, "Front" and "Rear" are modified by the XL and XR commands. A value of 100% results in the movement in the table. Between 0 and 100%, the Front/Rear positions are moved closer to Center. For negative values, Front and Rear are exchanged. For example, with an XL of -100%, in State 0, Tripod A on the left side would be moving Rear to Center, and Tripod B would be moving Front to Center.

5. When a horizontal servo is moving, its speed will be adjusted based on the Front/Rear pulse widths, the XL/XR percentage, and the XS percentage. Regardless of the travel distance from front to rear (adjusted by XL/XR), the total move time will be the HT divided by the XS percentage.
6. When a vertical servo is moving from Low to Mid or from Mid to Low, it will move at the speed specified by the VS command. When a vertical servo is moving from Mid to High or High to Mid, the vertical speed will be adjusted so that the horizontal and vertical movements end at the same time.
7. Any of the Hex Sequencer commands can be issued while the sequencer is operating. They will take effect immediately.

Hex Sequencer Examples:

"LH1000 LM1400 LL1800 RH2000 RM1600 RL1200 VS3000 <cr>"

Sets the vertical servo parameters.

"LF1700 LR1300 RF1300 RR1700 HT1500 <cr>"

Sets the horizontal servo parameters.

"XL50 XR100 XS100 <cr>"

Causes the gradual left turn at 100% speed (and starts the sequencer if it is not already started).

"XL -100 XR 100 XS 50 <cr>"

Causes a left rotate in place at 50% speed.

"XSTOP <cr>"

Stops the sequencer and allows servo channels 0-5, 16-21 to be controlled using the normal servo commands.

Query Hex Sequencer State.

XQ <cr>

Returns 1 digit representing the state of the hex sequencer, and the approximate percentage of movement in the state. The high nibble will be '0' to '7', and the low nibble will be '0' to '9'. For example, if the sequencer is 80% of the way through state 5, it will return the value 58 (hex).

Get Software Version.

VER <cr>

Returns the software version number as an ASCII string.

Firmware Upgrade.

Updating the firmware is an easy straight forward task, but can only be done at 115.2k baud. Unlike simple VER or servo move commands, updating the firmware requires fast bi-directional serial communications. If you are experiencing problems with this please check over the serial port troubleshooting guide to ensure your serial port and or USB to serial drivers are optimized.

Upgrading the firmware is best done with Lynxterm or one of our other software packages. Detailed instructions on upgrading the firmware is included in the software manuals. The "Force firmware update" jumper position (as illustrated in the Shorting Bar Jumpers and Connectors at a Glance section) is used when normal software firmware update will not work. Don't do this unless you know what you're doing.

Transfer to Boot.

GOBOOT <cr>

Starts the bootloader running for software updates. To exit the bootloader and start running the application, power cycle the control or enter (case sensitive, no spaces):

"g0000<cr>"

Mini SSC-II Emulation.

Binary format, 3-bytes.

Byte 1	255, the sync byte
Byte 2	0 - 31, the servo number
Byte 3	0 - 250, the pulse width; 0=500uS, 125=1500uS, 250=2500uS

SSC-32 Registers

Register General Info

Number	Name	Minimum	Maximum	Default	Description
0	Enable	0	65535	0	A bit field (16 bits) that enables various features of the SSC-32.
1	Transmit Delay	0	65535	600	The delay, in microseconds, before transmitting the first byte of a response from the SSC-32.
2	Transmit Pacing	0	65535	70	The delay, in microseconds, between bytes in a response from the SSC-32.
3-31	(Reserved)	--	--	--	--
32-63	Initial Pulse Offset	-100	100	0	The initial value of the pulse offset (PO) for each servo. Register 32 corresponds to servo #0, register 33 to servo #1, etc.
64-95	Initial Pulse Width	0	65535	1500	The initial value of the pulse width for each servo. Register 64 corresponds to servo #0, register 65 to servo #1, etc. A value of 0 leaves the servo output at a continuous logic '0'; a value of 65535 leaves the servo output at a continuous logic '1'. All other values are clipped to the range 500 - 2500 microseconds.
96-255	(Reserved)	--	--	--	--

Note: Registers 0-15 are intended for global use, affecting all operation of the SSC-32; registers 32-255 are intended for individual servo channel configuration, and so are in groups of 32 registers.

Enable Register (R0) Bit Definitions

Bit	Name	Definition
15 (msb)	Global Disable	If '1', disables all of the featured controlled by the Enable register. If '0', the individual bit values will be used to enable or disable the features.
14-4	(Reserved)	--
3	Initial Pulse Width Enable	If '1', enables the Initial Pulse Width register values at startup. If '0', the default value of 0 will be used.
2	Initial Pulse Offset Enable	If '1', enables the Initial Pulse Offset register values at startup. If '0', the default value of 0 will be used.
1	TX Delay/Pacing Enable	If '1', enables the Transmit Delay and Transmit Pacing values. If '0', the default values of 600uS and 70uS will be used.
0 (lsb)	Startup String Enable	If '1', enables execution of the startup string when power is applied to the SSC-32. If '0', the startup string will not be executed.

Register Read/Write

Command	Argument	Description	Examples
Register write: R <r> = <n> <cr>	r = reg number, 0-255 n = value	Programs the value of a register. Spaces are optional around the register number and value.	R0=1023 <cr> R32 = -50 <cr>
Register read: R <r> <cr>	r = reg number, 0-255	Displays the value of a register, followed by a carriage return. The displayed value is in ASCII format, and is terminated with a carriage return.	R0 <cr> result: 1023<cr> R32 <cr> result: -50<cr>
Set to defaults: RDFLT <cr>	none	Sets all of the registers to their default values. When the command is complete the SSC-32 will transmit the string OK<cr>.	RDFLT <cr> result: OK<cr>

The RDFLT command may take over a second to execute. It should not be invoked while a timed move or sequence player is active. No register writes should be performed until the RDFLT is complete (as indicated by the 'ok' response).

If multiple R= commands are being sent by software, it is recommended that the software read the value of each register after it is written. This will ensure that each register write has completed before the next is started.

If an RDFLT or R= command is executing, do not power down the SSC-32 until the command has completed. To determine whether the command has completed, read a register value.

Each time a register is written, the EEPROM location(s) used to store the value experience a small amount of wearout. The typical maximum number of writes is 100,000. Do not write your software to rapidly change the register values, or you might cause a permanent wearout of the EEPROM in the ATMega168 processor.

Miscellaneous Register Commands

Command	Argument	Description	Examples
STOP <n> <cr>	0-31	Immediately stops the specified servo at its current position. A space is optional before the servo number.	STOP0 <cr> STOP 31 <cr>

If the servo is part of a timed move, the other servos will continue moving and a query command will indicate that the move continues until the total time for the original move has elapsed. This is true even if *all* of the servos in the original move are stopped.

The EER and EEW commands no longer work to access internal EEPROM. They are replaced by Register Read/Write and Startup String commands. EER and EEW continue to function for external EEPROM.

Startup Strings

Command	Argument	Description / Examples
Delete characters: SSDEL <n> <cr>	0-255	Deletes <n> characters from the end of the startup string. If <n> is greater than the length of the startup string, then SSDEL deletes the entire string. SSDEL 5 <cr> - Deleted the last 5 characters of the startup string SSDEL 255 <cr> - Deletes the entire startup string
Concatenate: SSCAT <string> <cr>	Up to 100 ASCII characters	Concatenates <string> to the current startup string. The blank space immediately following "SSCAT" is ignored, but all other spaces are part of the string. The string is terminated by a carriage return, and may not contain embedded carriage returns. Commands in the startup string are terminated with a semicolon (including the last command). SSCAT #0P1000#1P2000T3000;<cr> SSCAT PL0 SQ5 SM50;<cr>
Display startup string: SS <cr>	none	Displays the entire startup string, surrounded by quotation marks and followed by a carriage return. SS <cr> result: "#0P1000#1P2000T3000;PL0 SQ5 SM50;"<cr>

The programmed startup string is executed at powerup of the SSC-32, if the Startup String Enable bit is set in the Enable register. The Startup String is executed after any register values are applied (e.g. initial pulse width).

The maximum total length of the startup string is 100 ASCII characters. Any additional characters will be ignored.

The following commands should not be used in a startup string: EER, EEW, R=, SSCAT, SSDEL.

The SS command may take hundreds of milliseconds to execute, depending on baud rate. It should not be invoked while a timed move or sequence player is active.

The SSCAT command may take hundreds of milliseconds to execute. It should not be invoked while a timed move or sequence player is active.

If an SSDEL or SSCAT command is executing, do not power down the SSC-32 until the command has been completed. To determine whether the command has completed, send an SS command and wait for the response.

Each time the startup string is changed the EEPROM locations used to store the value experience a small amount of wearout. The typical maximum number of writes is 100,000. Do not write your software to rapidly change the startup string, or you might cause permanent wearout of the EEPROM in the ATmega168 processor.

Startup String Examples	
Command	**Result**
SSDEL 255 <cr> SS <cr>	""<cr>
SSCAT #0P2000T5000;<cr> SS <cr>	"#0P2000T5000;"<cr>
SSCAT XXXX<cr> SSC <cr>	"#0P2000T5000;XXXX"<cr>
SSDEL 4 <cr> SS <cr>	"#0P2000T5000;"<cr>
SSDEL 6 <cr> SS <cr>	"#0P2000"<cr>
SSCAT #1P1000T4000;PL0SQ5;<cr> SS <cr>	"#0P2000#1P1000T4000;PL0SQ5;"<cr>

Additional Examples

Additional Examples	
Command	**Result**
RDFLT	Set all registers to default values

SSDEL 255	Erase the startup string
R0	Display register 0
R0=2 R1=2000 R2=1000	Set TX delay to 2000uS and TX pacing to 1000uS. (R0=2: Bit 1 of R0 enables TX delay/pacing.)
R0=12 R32=50 R64=1000	Set the pulse offset for servo 0 to 50 and the initial pulse width to 1000. (Bits 2 and 3 of R0 enable pulse offset and pulse width.)
R0=13 SSDEL 255 SSCAT #0P1500T5000;	Move R0 slowly to a pulse width of 1500 at startup. Assume the initial pulse width is set as in the previous example. (Bit 0 of R0 enables the startup string.)
SS	Display the current startup string.

Troubleshooting

Testing the Controller

The easiest way to test the controller is to use LynxTerm, our free terminal program. LynxTerm is downloadable here.

Once installed, click on the Port drop down and select your com port. This will work with USB-to-serial port adapters. Install the jumpers for 115.2k baud and the two DB9 serial port enable jumpers. Plug a straight-through DB9 M/F cable from the PC to the controller.

Install two servos, one on Channel 0 and the other on Channel 1.

Power up the SSC-32 (Logic and Servo) and notice the green LED is illuminated.

Click on the terminal window so you can type the following into it. Remember that <cr> means to press the Enter key.
#0 P1500 #1 P1500 <cr>

You should notice both servos are holding position in the center of their range. The LED is also no longer illuminated. It will now only light when the controller is receiving data. Type the following:
#0 P750 #1 P1000 T3000 <cr>

You should notice servo 0 moving CW slowly and servo 1 moving CCW a bit faster. They will arrive at their destinations at the same time even though they are moving different distances.

Now to test the Query Movement Status. Type the following:
#0 P750 <cr>

Then type the following line. This will make the servo move full range in 10 seconds.
#0 P2250 T10000 <cr>

While the servo is moving, type the following:
Q <cr>

When the servo is in motion the controller will return a "+". It will return a "." when it has reached its destination.

To experiment with the speed argument, try the following:
#0 P750 S1000 <cr>

This will move the servo from 2250 to 750 (around 170□) in 1.5 seconds.
Travel Distance / Speed Value = Travel Time
(2250uS-750uS) / (1000uS/Sec.) = 1.5 Sec.

Next try typing the following:
#0 P2250 S750 <cr>

This will move the servo from 750 to 2250 (around 170□) in 2.0 seconds.
(2250uS-750uS) / (750uS/Sec.) = 2.0 Sec.

Speed values above around 3500 will move the servo as quickly as it can move.

General Troubleshooting

If you notice the servos turn off or stop holding position when moving several servos at one time, this indicates that the SSC-32 has reset. This can be verified by noticing if the green LED is on steady after the servos are instructed to move. The green LED is not a power indicator, but a status indicator. When the SSC-32 is turned on, the LED will be on steady. It will remain on until it has received a serial character, then it will go out and only blink when receiving serial data.

The SSC-32 has two power supply inputs. The logic supply (VL) powers the microcontroller and its support circuitry through a 5vdc regulator. The servo supply (VS) powers the servos directly. In single supply mode (default) the jumpers VS1=VL will provide power to the 5vdc regulator from the VS terminal. This works great for battery use, and with most wall pack use, as long as the voltage does not drop too much. However, if it does drop, the voltage to the microcontroller is interrupted and the SSC-32 resets. To fix this you remove the VS1=VL jumper and connect a 9vdc battery clip to the VL input. This isolates the servo and logic supplies so one cannot affect the other.

Using the single supply mode is generally safe for the following conditions:

- VS of 7.2vdc 2800mAh NiCad or NiMH battery packs for up to 24 servos.
- VS of 7.4vdc 2800mAh LiPo battery packs for up to 24 servos.
- VS of 6.0vdc 1600mAh NiCad or NiMH battery packs for up to 18 servos.
- VS of 6.0vdc 2.0amp wall pack for up to 8 servos.

Note, these are just general guidelines and some exceptions may exist. The only other thing that can cause this effect is a poor power delivery system. If the wires carrying the current are too small, or connections are made with stripped and twisted wire, or cheap plastic battery holders are used, the same problem may occur. 99% of problems with the SSC-32 are power supply related. If you are noticing erratic or unstable servo movements, look at the power delivery system.

Communicating - USB to Serial Cables

The fastest way to use the SSC-32 is with a PC that has a real serial port. If you have a desktop PC that does not have a serial port it is recommended that you install a serial port card into the PC. Here is a mini tutorial showing the basics of adding a serial port. If you're using a laptop or PC without a serial port you can use a USB to serial cable. We recommend the FTDI USB to serial cable.

Note: FTDI based USB to serial cables have properties in their drivers called Latency and Buffer. You will need to reduce the size of the buffer from 4k to 1k, and the Latency from 16 to 1. As drivers and hardware change it may be necessary to experiment with these values.

If you have trouble with a USB to Serial cable, please refer to our troubleshooting guide.

Basic Atom Programming Examples

Atom / SSC-32 Test Example

```
' Atom / SSC-32 Test
' Configure the SSC-32 for 38.4k baud.

' Note, a | means the line continues onto the next line.
' Note, a ' means the line is a comment, and will not be compiled.

servo0pw var word
movetime var word

servo0pw = 1000
movetime = 2500

start:
```

```
servo0pw = 1000
serout p0,i38400,["#0P",DEC servo0pw,"T",DEC movetime,13]
pause 2500
servo0pw = 2000
serout p0,i38400,["#0P",DEC servo0pw,"T",DEC movetime,13]
pause 2500
goto start
```

Simple Biped Example

```
' Biped example program.
aa  var byte '<- general purpose variable.
rax var word '<- right ankle side-to-side. On pin0
ray var word '<- right ankle front-to-back. On pin1
rkn var word '<- right knee. On pin2
rhx var word '<- right hip front-to-back. On pin3
rhy var word '<- right hip side-to-side. On pin4
lax var word '<- left ankle side-to-side. On pin5
lay var word '<- left ankle front-to-back. On pin6
lkn var word '<- left knee. On pin7
lhx var word '<- left hip front-to-back. On pin8
lhy var word '<- left hip side-to-side. On pin9
ttm var word '<- time to take for the current move.

' First command to turn the servos on.
for aa=0 to 9
serout p0,i38400,["#", DEC2 aa\1, "P", DEC 1500, 13]
next

start:
' First position for step sequence, and time to move, put in your values here.
rax=1400: ray=1400: rkn=1400: rhx=1400: rhy=1400
lax=1400: lay=1400: lkn=1400: lhx=1400: lhy=1400
ttm=1000
gosub send_data
pause ttm

' Second position for step sequence, and time to move, put in your values here.
rax=1600: ray=1600: rkn=1600: rhx=1600: rhy=1600
lax=1600: lay=1600: lkn=1600: lhx=1600: lhy=1600
ttm=1000
gosub send_data
pause ttm

' Third...

' Forth...

' Etc...

goto start

' This sends the data to the SSC-32. The serout is all one line. Use |!
send_data:
serout p0,i38400,["#0P",DEC rax,"#1P",DEC ray,"#2P",DEC rkn,"#3P",DEC |
rhx,"#4P",DEC rhy,"#5P",DEC lax,"#6P",DEC lay,"#7P",DEC lkn,"#8P",DEC |
lhx,"#9P",DEC lhy,"T",DEC ttm,13]
return
```