Web Site: www.parallax.com
Forums: forums.parallax.com
Sales: sales@parallax.com
Technical: support@parallax.com

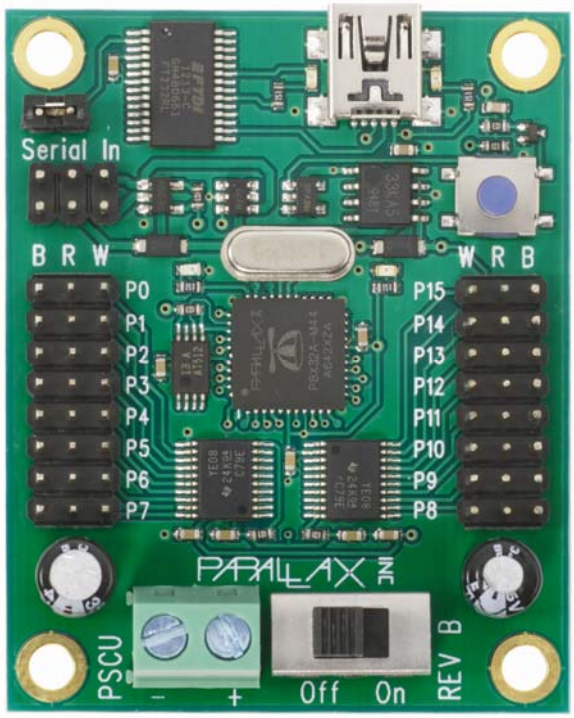
Office: (916) 624-8333
Fax: (916) 624-8003
Sales: (888) 512-1024
Tech Support: (888) 997-8267

Propeller Servo Controller USB (#28830)

The Propeller Servo Controller USB allows you to control up to 16 servos by sending serial commands from a microcontroller or PC via serial or USB connection. 32 servos can be controlled when using two Propeller Servo Controllers with a microcontroller. Since the Propeller Servo Controller USB is powered by a Propeller chip and the firmware is open-source, the Propeller Servo Controller USB is also a development platform which can be customized for specific applications. This could include systems that require servos, DC motors, stepper motors and lighting control.

Features

- Propeller P8X32A based hardware
- Compatible with previous Parallax servo controllers
- Separate screw-terminal power supply and power switch for servos
- Open-source firmware
- Program via USB interface with PC or serial interface to microcontroller
- Servo Ramping
- Network Ready – two units may be linked to control 32 servos (via microcontroller only)



Application Ideas

- Fun-house prop control
- Animated art control
- Robotics

Key Specifications

- Power requirements: 5 VDC @ ~60 mA for logic, 4.8 – 7.5 VDC for servos (do not exceed your servos' input voltage rating)
- Communication: Asynchronous Serial @ 2400 bps or 38.4 kbps (TTL or USB)
- Operating temperature: 32 to 158 °F (0 to 70 °C)
- Dimensions: 2.26 x 1.80 x 0.65 in (57.3 x 45.7 x 16.5 mm)

Before Connecting Your Propeller Servo Controller USB

- Programming the Propeller Servo Controller USB (PSCU) requires regulated 5 VDC. This can come from the USB port when used; (USB A to Mini B cable (#805-00006) required, sold separately). Or, supply 5 VDC to the center terminal on the Serial In header.
- Servos require a separate power supply connected to the screw terminal. Typically this needs to be from 4.8 – 7.5 VDC, and capable of supplying adequate current for the servos you are connecting. **Consult your servos' documentation for voltage limits and current requirements. Do not exceed 7.5 VDC input.**

Connecting and Testing

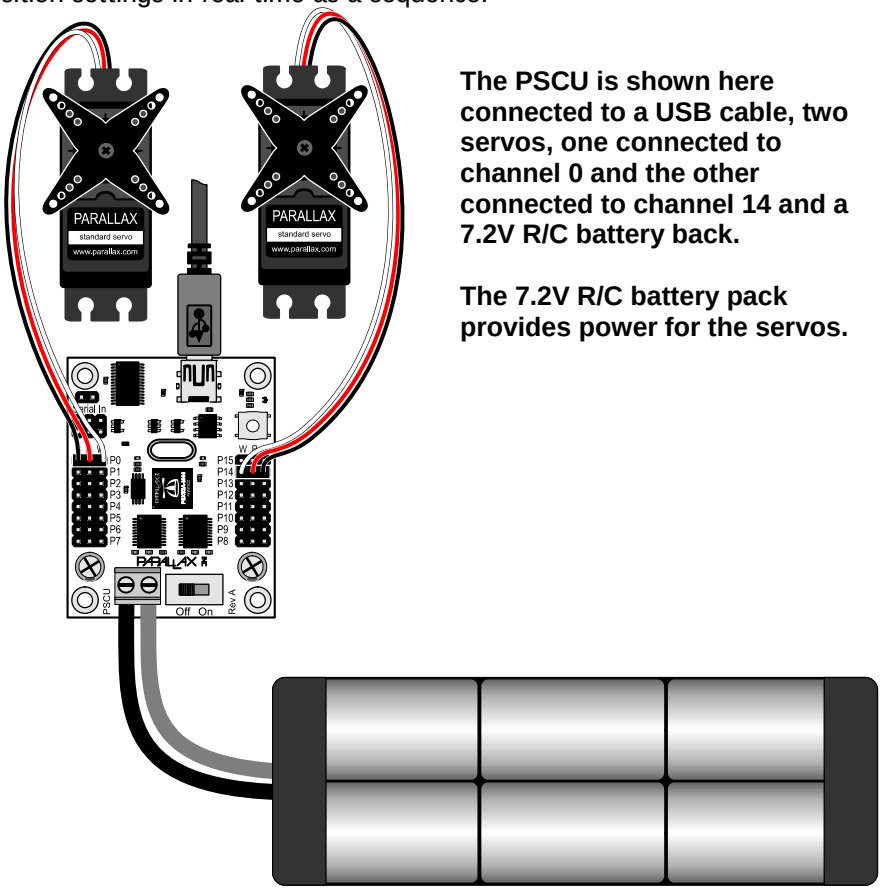
To get started using your new PSCU follow the directions on the following pages. Choose the type of connection you will be using: USB or TTL Serial via the "Serial In" connection, and then refer to those instructions below.

USB Connection

If you wish to control your PSCU from a PC using the USB port, then you will first need to install the FTDI VCP drivers. You may obtain the latest drivers for Windows by pointing your web browser to http://www.parallax.com/usbdrivers. You may also obtain drivers for Windows and other operating systems from from the FTDI website at http://www.ftdichip.com/Drivers/VCP.htm. Note: you may already have the USB Windows drivers installed if you have installed the BASIC Stamp Editor or Propeller Tool software or use any Parallax USB development boards.

Once the USB driver has been installed, you may connect your USB cable between the PC and the PSCU. You may also connect your servo power supply and servos as shown in the diagram below. The PSCU is now ready to receive serial commands from the host PC. Moving the power switch to the ON position will provide power to the servos. Note that it is not necessary to power on the servos for the PSCU to be ready for serial communication. As long as the USB port is providing power, the PSCU is powered on.

You can test your PSCU using the beta PSCI software listed on the PSCU product page on our website. This software will allow you to control servos by moving graphical sliders up and down, and serves as a quick way to verify operation. This software can also store positions for all sixteen channels and play back various position settings in real time as a sequence.



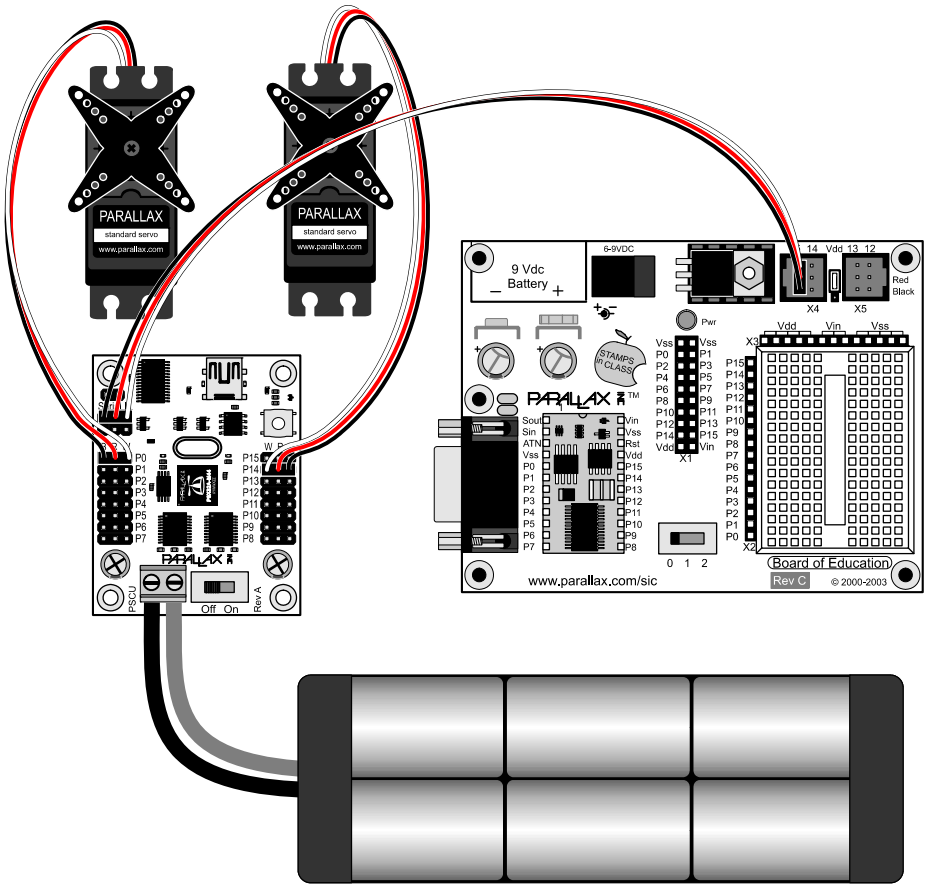
The PSCU is shown here connected to a USB cable, two servos, one connected to channel 0 and the other connected to channel 14 and a 7.2V R/C battery back.

The 7.2V R/C battery pack provides power for the servos.

TTL Serial Connection

If you wish to control your PSCU from a microcontroller then you will need to obtain a regulated 5 V from the host microcontroller, development board, or you may use a separate regulated 5 V power supply with a common ground. You also need to provide a TTL compatible serial signal to the PSCU. Ground, power and serial signal connections are made to the "Serial In" header as shown below. This 6-pin (2 x 3) header is actually two serial ports, one for incoming data and the other for outgoing data to another PSCU for networking. Notice that the connected cable is attached to the upper three pins with the black (ground) lead toward the outside. This upper port is the incoming serial connection. The lower three pins are used to connect to an additional PSCU for more channels.

You can test your PSCU using a BASIC Stamp 2, Board of Education and the code examples provided in the Command Set section of this documentation. Example programs for the BASIC Stamp demonstrate the use of each the commands and display responses for those commands which generate a reply.



The PSCU is shown here connected to a Board of Education, two servos, one connected to channel 0 and the other connected to channel 14 and a 7.2 V R/C battery back. The 7.2 V R/C battery pack provides power for the servos.

Communication Protocol

The PSCU supports several commands that are sent to it via serial protocol. These commands can come from the TTL serial interface or the USB port. Both serial inputs run at 2400 bps by default at startup and can be switched to 38.4 kbps by sending a command to the PSCU. The PSCU does not support auto-baud with the default firmware installed. The data must be sent non-inverted (true) using 8 data bits, no parity and 1 or 2 stop bits.

Command Set

Each command is preceded with an exclamation point (!) and the letters, "SC", so every command will appear as, "!SCxxxx" $0D, where, "!SC" is the preamble, "xxxx" are the command/parameter bytes and $0D is a trailing carriage return. Every command message contains exactly eight (8) bytes including the carriage return. All pulse width values are specified in 2 µs increments, so to send a 1 ms pulse to a channel you would use a pulse width value of 500 (500 x 2 µs = 1000 µs = 1 ms).

Position Command – Set Position of a Servo Channel

Syntax: "!SC" <channel> <ramp speed> <lowbyte> <highbyte> <CR>
Reply: None

To move a servo to a location you must write a position command to the PSCU. Each position command is comprised of the preamble, the channel, the ramp speed, lowbyte/highbyte of the pulse width in 2 µs increments and a carriage return ($0D). The preamble is "!SC". The channel is a byte value from 0 – 15 (16 – 31 if this is the second unit in a network). The ramp speed is a byte value from 0 – 63 that controls the speed the servo moves to its new position.

```
' {$STAMP BS2}
' {$PBASIC 2.5}

ch VAR Byte
pw VAR Word
ra VAR Byte

Sdat CON 15
baud CON 396

ra = 7
ch = 11

DO
  pw = 1100
  SEROUT Sdat, Baud+$8000,["!SC", ch, ra, pw.LOWBYTE, pw.HIGHBYTE, CR]
  PAUSE 1000
  pw = 300
  SEROUT Sdat, Baud+$8000,["!SC", ch, ra, pw.LOWBYTE, pw.HIGHBYTE, CR]
  PAUSE 1000
LOOP
```

Note: Not all servos have the same range limits. If your servos appear to strain when the pulse width is set to 250 you should increase this value up to 260 or higher as needed to prevent the servo from straining. Similarly, if your servo strains when the pulse width is set to 1250 you should reduce this value as needed to prevent the servo from straining.

VER? – Identify Firmware Version Number

Syntax: "!SCVER?" <CR>
Reply: "x.x"

The VER? Command causes the PSC to reply with its firmware version number. A carriage return ($0D) must follow the command. The version number returned is the version number of the firmware installed on the PSCU. The following code can be used to find and identify your PSCU using a BASIC Stamp 2.

```
' {$STAMP BS2}
' {$PBASIC 2.5}

Sdat PIN 15                             ' Serial Data I/O pin
Baud CON 396                            ' Constant for 2400 baud
buff VAR Byte(3)                        ' temporary variable

FindPSC:                                ' Find and get the version
  DEBUG "Finding PSC", CR               ' number of the PSC.
  SEROUT Sdat, Baud+$8000, ["!SCVER?",CR]
  SERIN Sdat, Baud, 500, FindPSC, [STR buff\3]
  DEBUG "PSC ver: ", buff(0), buff(1), buff(2), CR
```

Within the Debug Terminal you will see the message, "Finding PSC", followed by, "PSC ver:" and the firmware version (for example 1.0). If the message "Finding PSC" appears more than 3 or 4 times without a reply you should check to make sure all connections are correct and verify your power source. If the PSCU still fails to respond you may press the reset button momentarily. If the PSCU still fails to respond you may contact our Technical Support Department. Contact information is listed toward the end of this documentation.

SBR – Set the Baud Rate (to either 2400 or 38.4k Baud)

Syntax: "!SCSBR" <mode> <CR>
Reply: "BR" <mode>

After establishing communication with the PSCU you may wish to increase the baud rate to 38.4 kbps. The command for achieving this is shown above. You must first send "!SCSBR" followed by a byte value for the mode where 0 is for 2400 bps and 1 is for 38.4 kbps. Finally a carriage return ($0D) completes the command. The following code can be used to set the baud rate to 38.4 kbps using a BASIC Stamp 2.

```
' {$STAMP BS2}
' {$PBASIC 2.5}

Sdat PIN 15                           ' Serial Data I/O pin
Baud CON 396                          ' Constant for 2400 baud
buff VAR Byte(3)                      ' temporary variable

SetBaud:
  DEBUG "Setting Baudrate", CR
  SEROUT Sdat, Baud+$8000, ["!SCSBR",1,CR]
  SERIN Sdat, 6,500, SetBaud, [STR buff\3]
  DEBUG "Baud reply: ", buff(0), buff(1), DEC1 buff(2), CR
  STOP
```

Note that the SERIN command has a value of 6 for 38.4 kbps on the BASIC Stamp 2. This is because the PSCU will reply at the new baud rate to confirm that the command has been executed. If you would like to set the baud rate back to 2400 bps you must send a new SBR command at 38.4 kbps. This time the reply will be at 2400 bps so the SERIN will need to change as well. You can also reset the baud rate back to 2400 bps by pressing the reset button on the PSCU.

If either the BASIC Stamp or the PSC was reset without the other resetting they could be left in the state of different baud rates. When checking the firmware version using the VER? command, the BASIC Stamp 2 should employ its timeout feature. At the timeout label you can then attempt to VER? the PSCU at the other baud rate. Once identified the baud rate could then be set properly and the program can resume.

RSP – Report Servo Position

Syntax: "!SCRSP" <channel> <CR>
Reply: <channel> <highbyte> <lowbyte>

The RSP command returns the pulse width value last set for the specified channel. When the RSP command is sent to the PSCU it replies with three bytes. The first byte is the channel, the second and third bytes are the highbyte and lowbyte of the pulse width, respectively.

```
' {$STAMP BS2}
' {$PBASIC 2.5}

ch VAR Byte
pw VAR Word
ra VAR Byte
x VAR Byte

Buff VAR Byte(3)
Sdat CON 15
baud CON 396

Init:
  ra = 15: ch = 0

DO
  pw = 1100: GOSUB WRservo
  pw = 300: GOSUB WRservo
LOOP

WRservo:
  SEROUT Sdat, Baud+$8000,["!SC", ch, ra, pw.LOWBYTE, pw.HIGHBYTE, CR]
  FOR x = 0 TO 4
    PAUSE 1000
    SEROUT Sdat, Baud+$8000, ["!SCRSP", ch, CR]
    SERIN Sdat, Baud, 1000, Init,[STR Buff\3]
    DEBUG "Servo ", DEC buff(0), " ", HEX2 buff(1), " :", HEX2 buff(2), CR
  NEXT
  RETURN
```

Within the DO...LOOP, this program sets the pulse width (pw) to one extreme or the other and writes this value to the PSCU. The ramp value (ra) was set to give the program time to poll the servo position several times. Within the WRservo subroutine the new pw is sent and the position is polled five times, once per second. A DEBUG command is used to format the reply and print it to a window for you to view.

PSS – Set Software Port (to range 0 – 15 or 16 – 31)

Syntax: "!SCPSS" <mode> <CR>
Reply: "PM" <mode>

The PSS command assigns the PSCU to act on commands sent to channels 0 – 15 or channels 16 – 31 depending on the mode selected. This is useful when networking two PSCU modules together for 32 channels using a microcontroller. The command format is "!SCPSS" followed by a byte value of 0 for Port 0 (channels 0 – 15) or 1 for Port 1 (channels 16 – 31) and a carriage return ($0D).

```
' {$STAMP BS2}
' {$PBASIC 2.5}

Sdat            PIN     15              ' Serial Data I/O Pin
Baud            CON     396             ' Constant For 2400 Baud
buff            VAR     Byte(3)         ' Temporary Variable (Array)

SetPort:
  DEBUG "Setting Port Mode...", CR
  SEROUT Sdat, Baud+$8000, ["!SCPSS", 1, CR] ' Set To 16-31
  SERIN  Sdat, Baud,500, SetPort, [STR buff\3]
  DEBUG "Baud Reply:  ", buff(0), buff(1), DEC1 buff(2), CR
  DEBUG "Please reset PSCU or cycle power now."
  STOP
```

Note: you should be sure only the PSCU that you want to change port modes on is connected. Networking the PSCU modules while using this command will cause all connected units to be set. The PSCU must be reset after this command is sent or power must be cycled to activate the change.

PSD – Servo Disable

Syntax: "!SCPSD" <channel> <CR>
Reply: None

This command allows you to disable a servo channel. The command format is, "!SCPDS" followed by a byte value of 0 – 31 for the channel and a carriage return ($0D). Disabling a channel will cause the PSCU to stop sending pulses to that channel. This will cause any connected servo to become lax and not try to hold its position. A disabled servo channel can be enabled again using the Servo Enable command listed below. The following code for the BASIC Stamp 2 disables the servo on channel 0.

```
' {$STAMP BS2}
' {$PBASIC 2.5}

' {$STAMP BS2}
' {$PBASIC 2.5}

Sdat            PIN     15              ' Serial Data I/O Pin
Baud            CON     396             ' Constant For 2400 Baud

Disable:
  DEBUG "Disabling Channel 0", CR
  SEROUT Sdat, Baud+$8000, ["!SCPSD", 0, CR]
  STOP
```

Note: This command causes this to be the default state of this channel. Until the channel is enabled using the Servo Enable command (below) this channel will no longer refresh even if the PSCU is reset or power is cycled.

PSE – Servo Enable

Syntax: "!SCPSE" <channel> <CR>
Reply: None

This command allows you to enable a servo channel that has been previously disabled. The command format is, "!SCPSE" followed by a byte value of 0 – 31 for the channel and a carriage return ($0D). Enabling a channel will cause it to move to the last position it was commanded to, or the startup default if no other position commands have been sent since power up/reset. The following code (re)enables channel 0.

```
' {$STAMP BS2}
' {$PBASIC 2.5}

Sdat            PIN     15              ' Serial Data I/O Pin
Baud            CON     396             ' Constant For 2400 Baud

Enable:
  DEBUG "Enabling Channel 0", CR
  SEROUT Sdat, Baud+$8000, ["!SCPSE", 0, CR]
  STOP
```

EDD – Startup Servo Mode

Syntax: "!SCEDD" <mode> <CR>
Reply: "DL" <mode>

This command sets whether the PSCU centers all servo channels on startup (mode 0) or uses custom startup positions stored in EEPROM (mode 1). In mode 1 you can set a custom startup position for each servo channel using the default position command below. To set the Startup Servo Mode, the following must be sent to the PSCU: "!SCEDD" followed by a byte value of 0 for default (center) or 1 for custom and finally a carriage return ($0D). The following code can be used to set the startup servo mode to 1 using a BASIC Stamp 2.

```
' {$STAMP BS2}
' {$PBASIC 2.5}

Sdat            PIN     15              ' Serial Data I/O Pin
Baud            CON     396             ' Constant For 2400 Baud
buff            VAR     Byte(3)         ' Temporary Variable (Array)

SetPort:
  DEBUG "Setting Startup Servo Mode...", CR
  SEROUT Sdat, Baud+$8000, ["!SCPSS", 1, CR]
  SERIN  Sdat, Baud,500, SetPort, [STR buff\3]
  DEBUG "Startup Mode:  ", buff(0), buff(1), DEC1 buff(2), CR
  DEBUG "Please reset PSCU or cycle power now."
  STOP
```

Note: The PSCU must be reset after this command is sent or power must be cycled to activate the change.

Default Position Command – Set Default Position of a Servo Channel

Syntax: "!SCD" <channel> <lowbyte> <highbyte> <CR>
Reply: None

By default all servos move to the center position at startup (they receive a 1.5 ms pulse). The default position of all servo channels can be customized and stored in the EEPROM to be used on startup in place of the center positions. To set a new startup position for a servo channel the following must be sent to the PSCU: "!SCD" followed by a byte value of 0 – 31 for channel, two bytes for the pulse width in 2 µs units in lowbyte/highbyte format, and finally a carriage return ($0D). Each servo channel can be set independently but for the PSCU to use these on startup you must have first sent the EDD command (listed above) to 1. The following code can be used to set the default position for channels 0 through 31 to 500 (1 ms) using a BASIC Stamp 2. If you only have one PSCU connected only channels 0 – 15 will be affected by this program.

```
' {$STAMP BS2}
' {$PBASIC 2.5}

ch              VAR     Byte            ' Channel
pw              VAR     Word            ' Pulse Width Value
Sdat            PIN     15              ' Serial Data I/O Pin
Baud            CON     396             ' Constant For 2400 Baud
pw = 500

Set_Default:
  DEBUG "Setting Default Position...", CR
  FOR ch = 0 TO 31
    SEROUT Sdat, Baud+$8000, ["!SCD", ch, pw.LOWBYTE, pw.HIGHBYTE, CR]
    DEBUG "Channel ", DEC ch, " set to ", DEC pw, CR
  NEXT
  DEBUG "Please reset PSCU or cycle power now.", CR
  DEBUG "Be sure the Start Servo Mode is set to 1"
  STOP
```

The PSCU must be reset after this command is sent or power must be cycled to activate the change.

CLEAR – Clear Upper EEPROM

Syntax: "!SCLEAR" <channel> <lowbyte> <highbyte> <CR>
Reply: "CLR"

All of the custom settings such as Port Mode, Servo Disabled, Startup Mode and Default Positions are all stored in the upper 32K of the EEPROM. The code below can be used to clear the upper EEPROM and reset these values to their defaults.

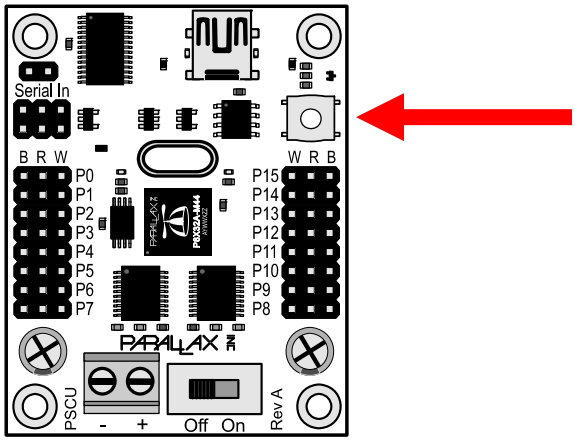
```
' {$STAMP BS2}
' {$PBASIC 2.5}

Sdat            PIN     15              ' Serial Data I/O Pin
Baud            CON     396             ' Constant For 2400 Baud
buff            VAR     Byte(3)         ' Temporary Variable (Array)

Clear_EEPROM:
  DEBUG "Clearing Upper EEPROM...", CR
  SEROUT Sdat, Baud+$8000, ["!SCLEAR", CR]
  SERIN  Sdat, Baud, 600, Clear_EEPROM, [STR buff\3]
  DEBUG  "Reply: ", buff(0), buff(1), buff(2), CR
  STOP
```

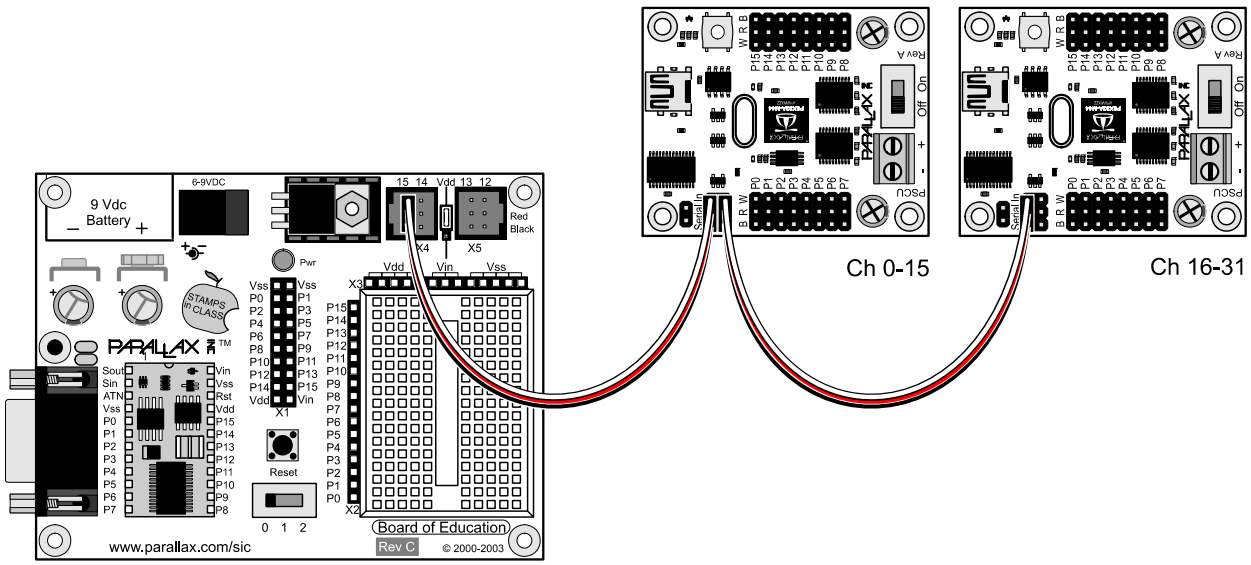
Hardware Reset Button

Some of the program examples above require the PSCU to be reset or power cycled before the settings will take effect. Pressing the reset button will restart the firmware, effectively resetting the PSCU.



Networking Two PSCU Modules for 32 Channels (Microcontroller)

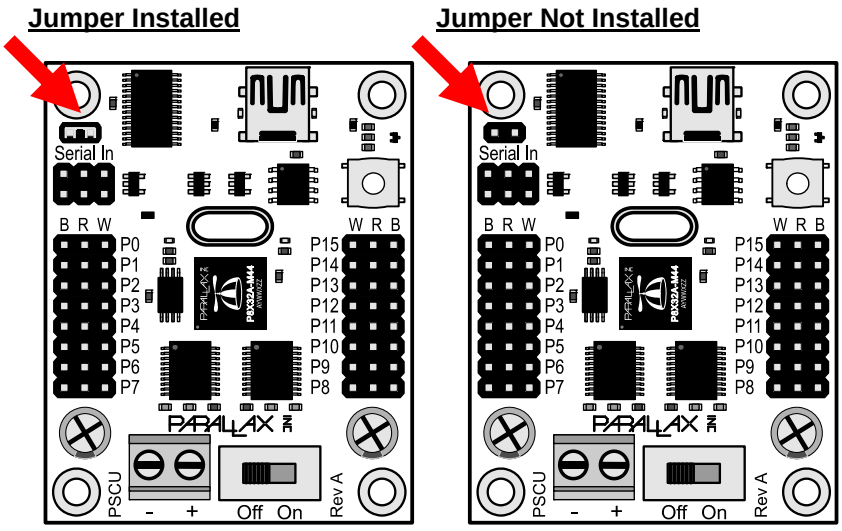
The PSCU is capable of networking to a second PSCU to allow control of 32 servos from a single serial line when using a microcontroller. The second PSCU would have its Port Mode set to 1 and would respond to commands sent to servo channels 16 – 31. A typical connection for doing this using the Board of Education and two PSCU modules is shown below. Note the positions of the cables in the serial connectors on each PSCU. Both PSCU modules will get power from the Board of Education in this manner and both will share the serial I/O line as well. Both PSCU modules will still require a separate power supply for the servos.



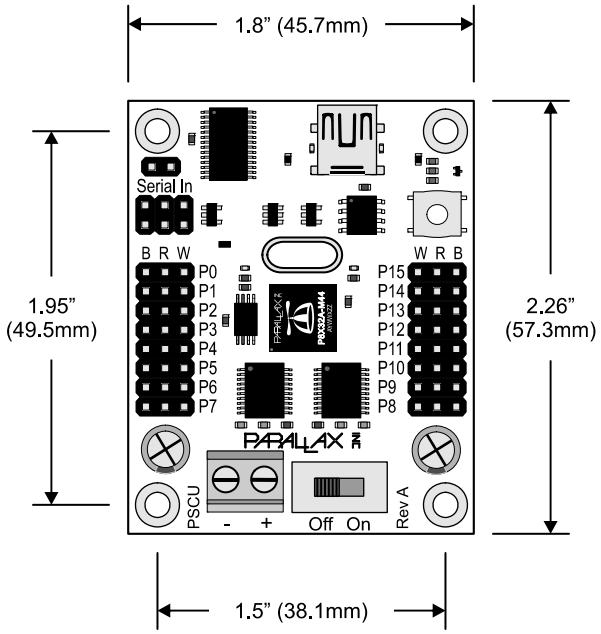
Note: Networking/Daisy-chaining of PSCU modules is not supported when using a USB connection. For USB control you will need a separate USB port for each unit and all units will be set to Port Mode 0.

Programming Jumper

The firmware on the PSCU can be customized by the end user. This makes it easy to tailor the use of this Servo Controller to your application's needs. You may also write you own firmware from scratch and take advantage of additional features of the hardware that may be implemented. If you wish to develop your own firmware you may simply download the Propeller Tool software from our website at http://www.parallax.com/propeller. The programming jumper is installed by default, but can be removed to prevent accidentally overwriting the code. Removing the jumper disconnects the DTR line from the /RES line on the Propeller, thereby disabling the Propeller chip from being programmed or identified. You can still communicate serially whether the jumper is installed or not.



Module Dimensions



Revision History

Version 1.1: Updated photo to show Rev B of the product. Manufacturing requirements for the QFN Propeller IC necessitated moving a via (pad) on the PCB. Three traces were also moved slightly. This is a manufacturing change only and the schematic has not changed.