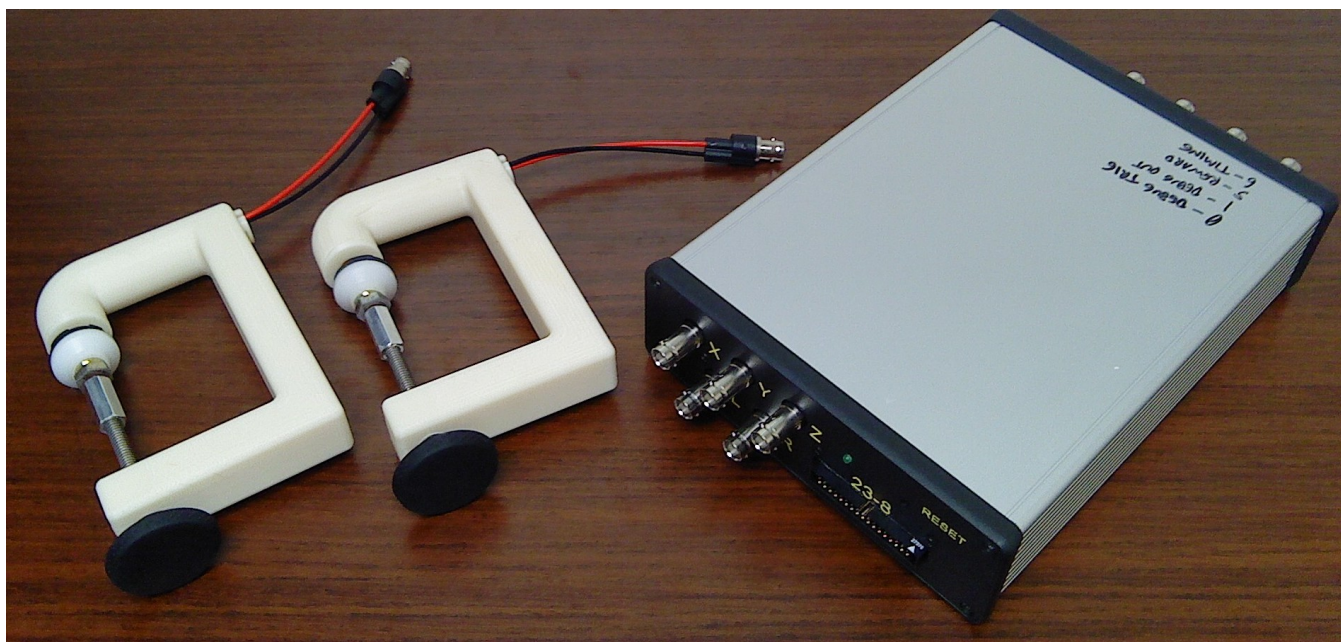


I/O SynchBox User Guide

Written by Christopher Thomas – February 19, 2021.



Contents

1	Overview	1
2	Command Interface	4
3	Accessories	10
4	Maintenance	11
4.1	Using avrdude On Linux, Windows, and MacOS	12

Chapter 1

Overview

The I/O SynchBox is a device that lets a host computer and several instruments talk to each other. It was commissioned by the Attention Circuits Control Laboratory to facilitate their experiments¹. A system diagram of the York University installation is shown in Figure 1.1. A diagram of the I/O signals used by the I/O SynchBox is shown in Figure 1.2.

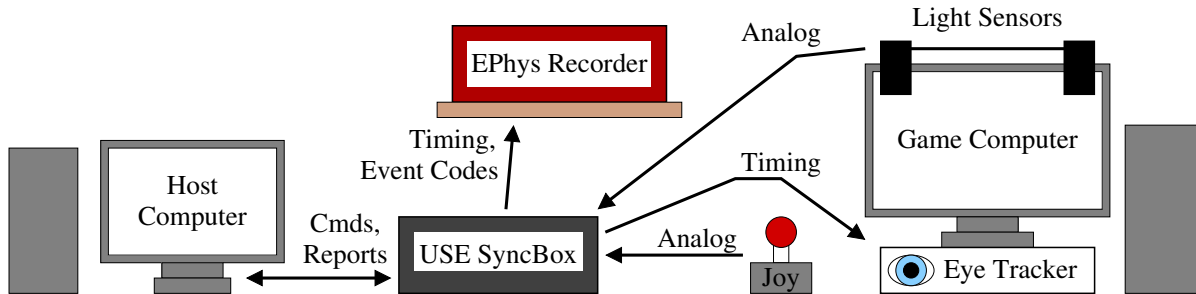


Figure 1.1: System block diagram.

¹I/O SynchBox integration is described in: “USE: An integrative suite for temporally-precise psychophysical experiments in virtual environments”, Watson, M. R., Voloh, B., Thomas, C. J., Hasan, A. M., and Womelsdorf, T. (2018), *bioRxiv*, 434944 doi:10.1101/434944

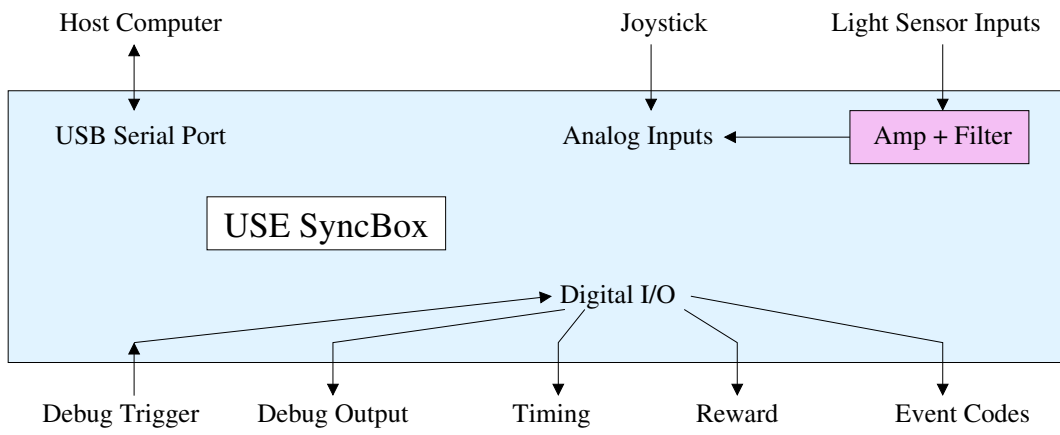


Figure 1.2: I/O SyncBox inputs and outputs.

The I/O SynchBox handles several types of communication:

- It talks to the host computer over a USB serial port connection.
- It generates timing pulses. These are controlled using the **Txx** series of commands. These are typically used to provide events with known timestamps to equipment such as eye-trackers and electrophysiology instruments.
- It generates single pulses of variable duration. These are controlled using the **RWD** command. These are typically used for dispensing a reward to test animals.
- It emits binary number values over a parallel digital interface. These are controlled using the **Nxx** series of commands. These are typically used as event codes for electrophysiology equipment.
- It reads analog information from three general-purpose analog inputs, and shows this data during logging. These are typically used for joystick inputs.
- It reads light sensors to synchronize with the game computer's display. The light sensor inputs have hardware amplification and filtering (they are special-purpose inputs, not general analog inputs). Light levels are shown during logging.
- Logged data is controlled using the **Lxx** series of commands.

The commands are described in detail in Section 2.

To started, connect to the I/O SynchBox (USB serial link at 115200 baud, 8N1), and type “?” (and enter) for help. Type “**QRY**” to see the I/O SynchBox 's settings.

The front and back panels of the I/O SynchBox are shown in Figure 1.3. The pinout of the parallel output is shown in Figure 1.4.



Figure 1.3: I/O SynchBox front and back panels.

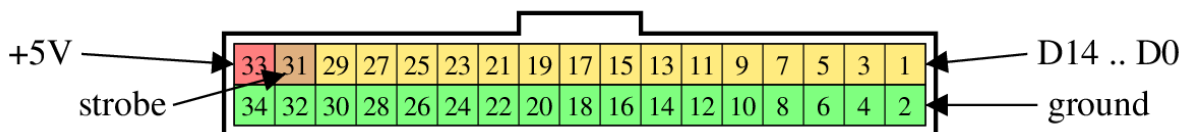


Figure 1.4: Event code connector pinout.

Chapter 2

Command Interface

The I/O SynchBox is connected to the host computer via a USB serial link at 115200 baud. The host issues commands, and the I/O SynchBox reports events that occur.

The I/O SynchBox issues timing pulses at regular intervals; issues reward pulses as one-off events; writes parallel event code data to the ephys I/O port; and reports analog input from the joystick and light sensors at regular intervals.

In addition to analog light sensor values, the I/O SynchBox also reports whether each sensor is above- or below-threshold. Light sensor thresholds are calibrated by measuring sensor values during a time window, taking the average of these measured values, and then saving the average as the new threshold (overwriting the old threshold). “Free-running” calibration does this repeatedly (starting a new window when the old window ends).

A list of available commands is shown in Figure 2.1. A timing diagram of user-commanded output events is shown in Figure 2.2. A diagram illustrating light sensor filtering is shown in Figure 2.3.

Sample output (including echoed commands) is shown in Figure 2.4. Format details for verbose, terse, and packed log entries are given in Figure 2.5. Typical configuration/status information is shown in Figure 2.6.

There are several important things to note about interactions with the I/O SynchBox :

- The command parser does **not** recognize backspaces.
- **All times**, whether they’re timestamps or duration arguments, are given in “**clock ticks**”, **not milliseconds**. By default, clock ticks are 0.1 ms in length. One second is 10000 ticks.

Milliseconds would not have been sufficiently precise, and microseconds would have overflowed a 32-bit counter in slightly more than one hour.

Bear in mind that the maximum argument value is 65535 (a duration or interval of about 6.5 seconds).

Commands:

?, HLP : Help screen.
QRY : Query system state.
ECH 1/0: Start/stop echoing typed characters back to the host.
IDQ : Device identification string query.
INI : Reinitialize (clock reset and events idled).
LOG 1/0: Start/stop reporting log data.
LIN n: Set the data reporting interval to n ticks.
LVB 2/1/0: Set data report verbosity (2 = full, 1 = terse, 0 = packed hex).
TPW n: Set the timing pulse duration to n ticks.
TPP n: Set the timing pulse period to n ticks.
TIM 1/0: Enable/disable timing pulses.
TBW n: Set the timing channel 2 pulse duration to n ticks.
TBP n: Set the timing channel 2 pulse period to n ticks.
TIB 1/0: Enable/disable timing channel 2 pulses.
RWD n: Send a reward pulse lasting n ticks.
RWB n: Send a reward channel 2 pulse lasting n ticks.
NSU n: Set the event code pre-strobe setup time to n ticks.
NHD n: Set the event code post-strobe hold time to n ticks.
NSE 1/0: Enable/disable strobing the most significant event code bit.
NPD n: Set the event code strobe pulse duration to n ticks.
NEU n: Emit value n over the event code parallel interface.
NDW n: Set event code data width to n bits (8 or 16).
CSL/R/I : Light sensor calibration slaved to left, right, or independent.
CAO/F n: Calibrate light sensors over n-tick window (one-shot/free-running).
CTR/L n: Force left/right light sensor threshold value to n.
FIL n: Set analog noise filter window to 2^n samples (0 disables).
FST n: Set analog spike rejection threshold to 2^n times the variance.
FSW n: Set spike rejection variance window to 2^n samples (0 disables).
Debugging commands:
XAL 1/0: Start/stop streaming full-rate analog light sample data.
XJL 1/0: Swaps or un-swaps joystick XY and light sensor LR inputs.
XTR 1/0: Enable/disable reward triggered by rising edge of GP00.
XTT 1/0: Enable/disable timing active on high level on GP00.
XTN 1/0: Enable/disable event code triggered by rising edge of GP00.
XMA n: Emit a rising edge on GP01 when analog 0..4 (X/Y/Z/L/R) rises.
XMD : Disable GP01 analog diagnostic output.

Figure 2.1: Command summary.

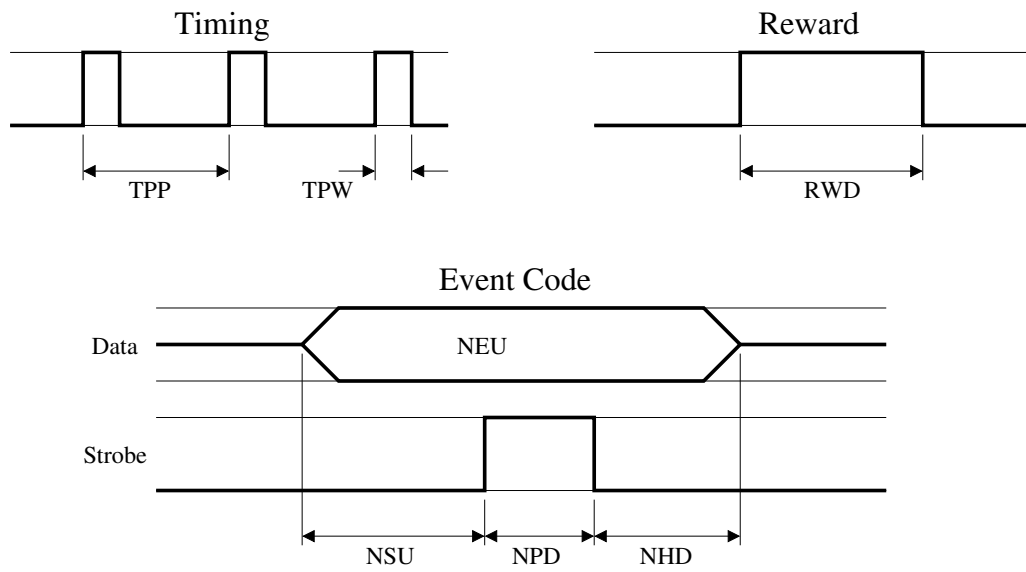


Figure 2.2: Output signal timing diagram.

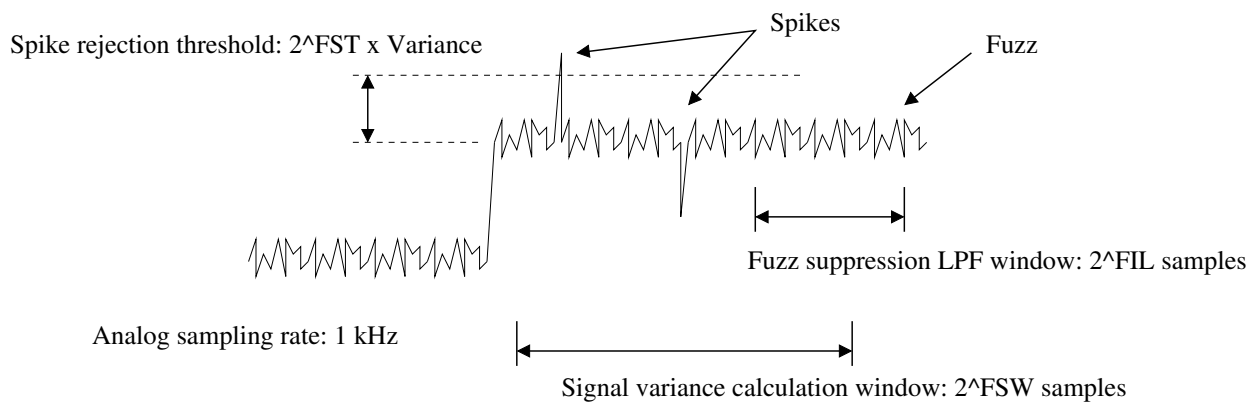


Figure 2.3: Light sensor filtering.

- Number values entered by the user are given in **decimal**, but number values reported from the I/O SynchBox are in **hexadecimal** (with the exception of status information).

Processing user information in decimal made the command parser simpler (and makes life easier for manually entering data). Reporting information in hexadecimal makes output more compact (important for high data rate output), and avoids time-consuming data conversion (converting to hexadecimal does not require divisions, only bit-shifts; converting to decimal does require divisions). The data rate is high enough that this matters.

The data reporting rate is limited by the serial connection. If events occur too fast to report (or if full-rate analog debug reporting exceeds the serial connection's ability to report), reports will be dropped. Events themselves will still occur.

At 115200 baud, the maximum reporting rate without packet loss is 150 Hz with verbose logging, 200 Hz with terse logging, and 300 Hz with packed hex logging.

```

log 1
Time: 0005764f Joy (x/y/c): 0c 00 00 Opt (l/r): 76 75 BLK BLK
Time: 00057a37 Joy (x/y/c): 0f 00 00 Opt (l/r): 76 76 BLK BLK
Time: 00057e1f Joy (x/y/c): 0e 00 00 Opt (l/r): 76 76 BLK BLK
lvb 1
T: 000631d0 XYC: 0d 00 00 LR: 76 76 B B
T: 000635b8 XYC: 0f 00 00 LR: 76 76 B B
T: 0006399f XYC: 0e 00 00 LR: 76 76 B B
lvb 0
T0006ddb0J0f0000L7676BB
T0006e198J0e0000L7676BB
T0006e57fJ0e0000L7676BB
lvb 2
Time: 000785a7 Joy (x/y/c): 0c 00 00 Opt (l/r): 76 76 BLK BLK
Time: 0007898f Joy (x/y/c): 0d 00 00 Opt (l/r): 76 76 BLK BLK
Time: 00078d77 Joy (x/y/c): 0f 00 00 Opt (l/r): 76 76 BLK BLK
log 0

tim 1
Synch: 00087e8e
Synch: 0008a59e
Synch: 0008ccae
tim 0

rwd 1000
Reward: 000a3433 03e8
rwd 1000
Reward: 000aa549 03e8
rwd 500
Reward: 000ba7ee 01f4

xal 1
99387676
99457675
99527675
995e7675
xal 0

neu 12345
Code: 000f2e19 3039
neu 32767
Code: 00101097 7fff

```

Figure 2.4: Sample I/O SynchBox output.

Verbose:

Time: 0003f6ef **Joy (x/y/c):** 062a 063f 0647 **Opt (l/r):** 0610 061b **BLK** **BLK**

Terse:

T: 0004d853 **XYC:** 055f 058f 05b3 **LR:** 05b0 056c **B** **B**

Packed:

T 00054937 **J** 05b8 05c2 05c5 **L** 056f 05cb **BB**

Figure 2.5: Log entry format details.

devicetype: USE SyncBox subtype: v1 revision: 20200128

System state (all values in base 10):

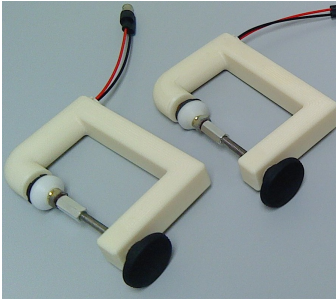
Version: 20200128
Timestamp: 37580 ticks
Clock ticks per second: 10000
Reward I/O: D11/B5/GP05 Reward ch2 I/O: D10/B4/GP04
Timing I/O: D12/B6/GP06 Timing ch2 I/O: D13/B7/GP07
Debug trigger input: D6/H3/GP00
Debug monitor output: D7/H4/GP01
Logging: OFF
Log interval: 100 ticks
Verbosity: Full
Timing: OFF
Timing pulse width/period: 10 / 10000 ticks
Timing ch2: OFF
Timing ch2 pulse width/period: 10 / 10000 ticks
Event code data front-porch/strobe/back-porch: 2 / 2 / 2 ticks
Event code strobe: ON
Event code data width: 16 bits
Light sensor thresholds (left/right): 65535 / 65535
Light threshold slaving: INDEPENDENT
Analog sample size: 8 bits
Noise suppression filter: 2 bits (decay length 4 samples)
Spike suppression filter:
Variance LPF: 6 bits (decay length 64 samples)
Spike rejection threshold: 2² (4x)
External event triggers: none
Analog input being monitored: none
Joystick analog threshold: 127
End of system state.

Figure 2.6: I/O SynchBox identification string and status report.

Chapter 3

Accessories

Accessories produced for the I/O SynchBox include the following:



Light sensor clamps may be attached to a wide variety of monitors (including notebook computers). Patches of the display are strobed, and this flickering is recorded in order to measure when frame redraws occur.

Chapter 4

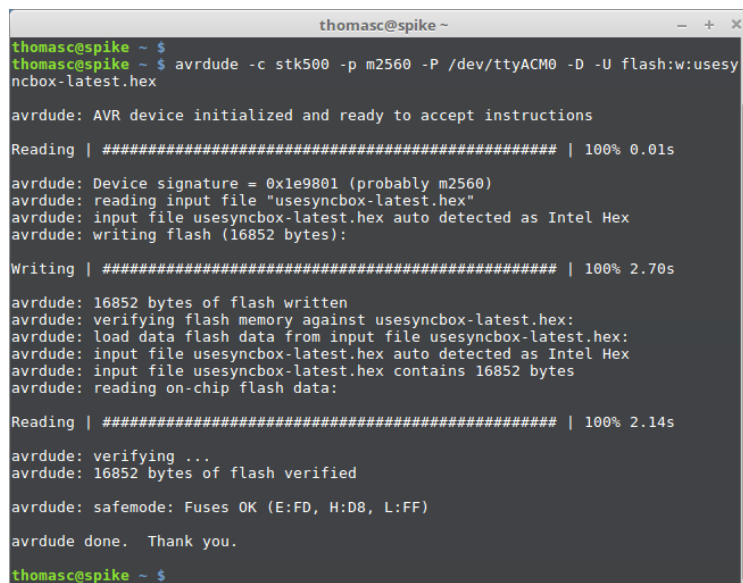
Maintenance

From time to time, new versions of the I/O SynchBox firmware are released. These can be flashed using the “avrdude” utility, which is bundled with the Arduino development environment or may be downloaded on its own.

To use “avrdude”, plug in the I/O SynchBox, determine what the name of the Arduino’s USB serial device is, bring up a “command terminal” window, navigate to the directory with the firmware file, and type:

```
avrdude -c stk500 -p m2560 -P (device) -D -U flash:w:(firmware file)
```

For a firmware file named “usesynchbox-latest.hex”, and a USB serial device called “/dev/ttyACM0”, the result will be similar to the following:



```
thomasc@spike ~  
thomasc@spike ~ $ avrdude -c stk500 -p m2560 -P /dev/ttyACM0 -D -U flash:w:usesynchbox-latest.hex  
avrdude: AVR device initialized and ready to accept instructions  
Reading | ##### | 100% 0.01s  
avrdude: Device signature = 0x1e9801 (probably m2560)  
avrdude: reading input file "usesynchbox-latest.hex"  
avrdude: input file usesynchbox-latest.hex auto detected as Intel Hex  
avrdude: writing flash (16852 bytes):  
Writing | ##### | 100% 2.70s  
avrdude: 16852 bytes of flash written  
avrdude: verifying flash memory against usesynchbox-latest.hex:  
avrdude: load data flash data from input file usesynchbox-latest.hex:  
avrdude: input file usesynchbox-latest.hex auto detected as Intel Hex  
avrdude: input file usesynchbox-latest.hex contains 16852 bytes  
avrdude: reading on-chip flash data:  
Reading | ##### | 100% 2.14s  
avrdude: verifying ...  
avrdude: 16852 bytes of flash verified  
avrdude: safemode: Fuses OK (E:FD, H:D8, L:FF)  
avrdude done. Thank you.  
thomasc@spike ~ $
```

The names of USB serial devices vary widely from operating system to operating system and version to version. Consult appropriate documentation to determine how to find the I/O SynchBox’s USB serial device on your operating system.

4.1 Using avrdude On Linux, Windows, and MacOS

While `avrdude` can be installed as a stand-alone package, an easy and reliable way to install it is to install the “Arduino” IDE (found at: <https://www.arduino.cc>). This will install several things:

- The Arduino development environment.
- The `avr-gcc` compiler and its libraries.
- The `avr-binutils` toolchain used with `avr-gcc`.
- The `avrdude` program for flashing AVR-based boards.

Various quirks show up on each operating system:

- Under Linux, you may have to add a “udev rules” file in `/etc/udev/` to get Arduino boards detecting and connecting properly. The Arduino IDE package *should* set this up for you.
- Under Windows 10, neither `avrdude` nor its configuration file will be on path by default. To use it, first find out where it was installed, and then either type the full path name ahead of the command or add the relevant directory to the path.
 - To find out where `avrdude` was installed, look for an “Arduino” shortcut on the desktop. Right-click it to edit its properties; this will tell you what directory the Arduino binary is in. Open the top-level directory in that tree, and tell Windows to search that folder to find “`avrdude`”. This will bring up the directory with `avrdude` in it, which should also contain the “`avrdude.conf`” configuration file.
 - To set the path in Windows 10, go to “control panel”, “system”, “advanced system settings”, “advanced” tab, “environment variables”. Under “user variables”, select “path” and click “edit”. Add the directory as a new entry in the list.
- To run `avrdude` from the command line under Windows, type the following (as one line):
`(avrdude path)/avrdude -C (avrdude path)/avrdude.conf -c stk500 -p m2560 -P (serial port) -D -U flash:w:(firmware file including its path)`
- Under MacOS, **FIXME: Information goes here.**