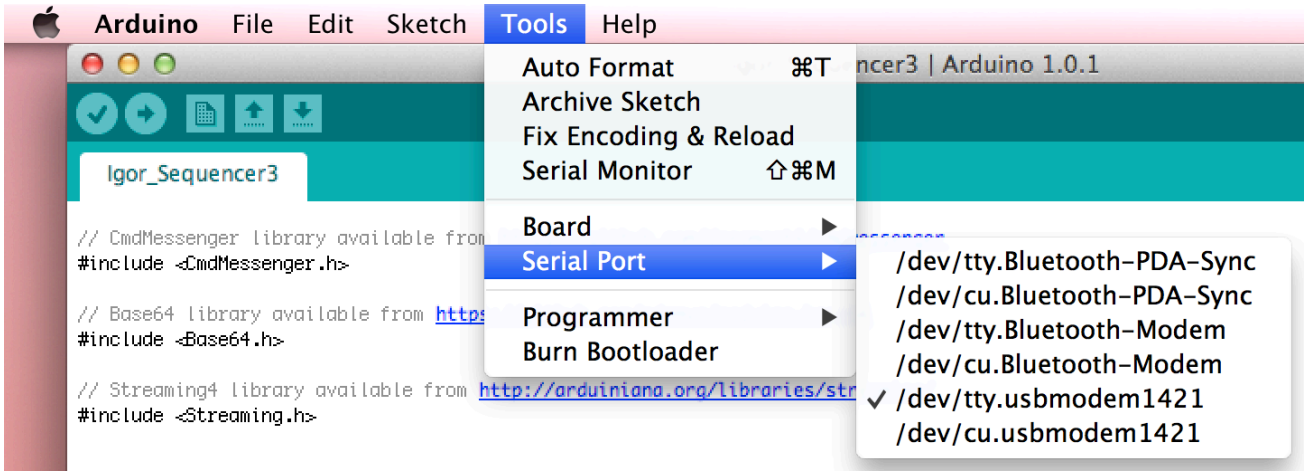
Igor Sequencer for Arduino

Igor Sequencer controls an Arduino device to produce sequences of output pulses in a user-defined sequence. It provides low cost control of up to 12 outputs in two banks of 6 pins with sub-millisecond accuracy. Sequences are programmed through a GUI and they can be saved and loaded for quick recall.

Arduino Installation

Install the Arduino and software for Windows or MacOS which can be downloaded from http://arduino.cc/en/Main/Software

Follow the installation instructions carefully and ensure that your Arduino is recognised. There are several example sketches, which you can compile and run, to ensure that the Arduino is connected and responding properly. Make a note of which Port the Arduino is connected to, as you will need to select this in Igor. This can be found from within the Arduino software under the heading Tools, Serial Port.



The Igor_Sequencer Sketch requires installation and inclusion of the following Arduino libraries

- CmdMessenger library available from https://github.com/dreamcat4/cmdmessenger
- Base64 library available from https://github.com/adamvr/arduino-base64
- Streaming5 library available from http://arduiniana.org/libraries/streaming/

Each of these libraries need to be placed in folders within the Arduino:Libraries folder. In my MacOS installation, this is in Documents:Arduino:Libraries:

The Igor_Sequencer3.ico sketch can be installed in Arduino:Igor_Sequencer3:Igor_Sequencer3.ico

On Windows and MacOS systems, the Arduino.exe file or Arduino application can run from any location but you need to maintain the same relative subfolder structure for the various include files. These are already included in the Igor_Sequencer3.ico sketch but it won't compile if the libraries are not present.

You should open up Igor_Sequencer3.ico in the Arduino software and choose the correct device and port. As an example on my Retina Powerbook (see above), it should be connected to /dev/tty.usbmodem1421 (right hand side USB connector) or /dev/tty.usbmodem1411 (left hand side USB connector). Make a note of this because you will need to ensure that you tell Igor to which port the Arduino is connected.

I have tested this with an Arduino Uno and an Arduino Mega 2560. The Mega 2560 is overkill for this particular application. I expect it will work with other supported Arduino devices provided they have sufficient memory. If all is well, the software should compile and can be loaded into the Arduino. If this works correctly, the LED attached to pin 13 should blink on and off at 2 second intervals. This was done to help establish that the device has loaded the code properly and that it is functioning but this "feature" it can be disabled in the sketch if you wish to use pin 13 for something more useful.

Igor Installation

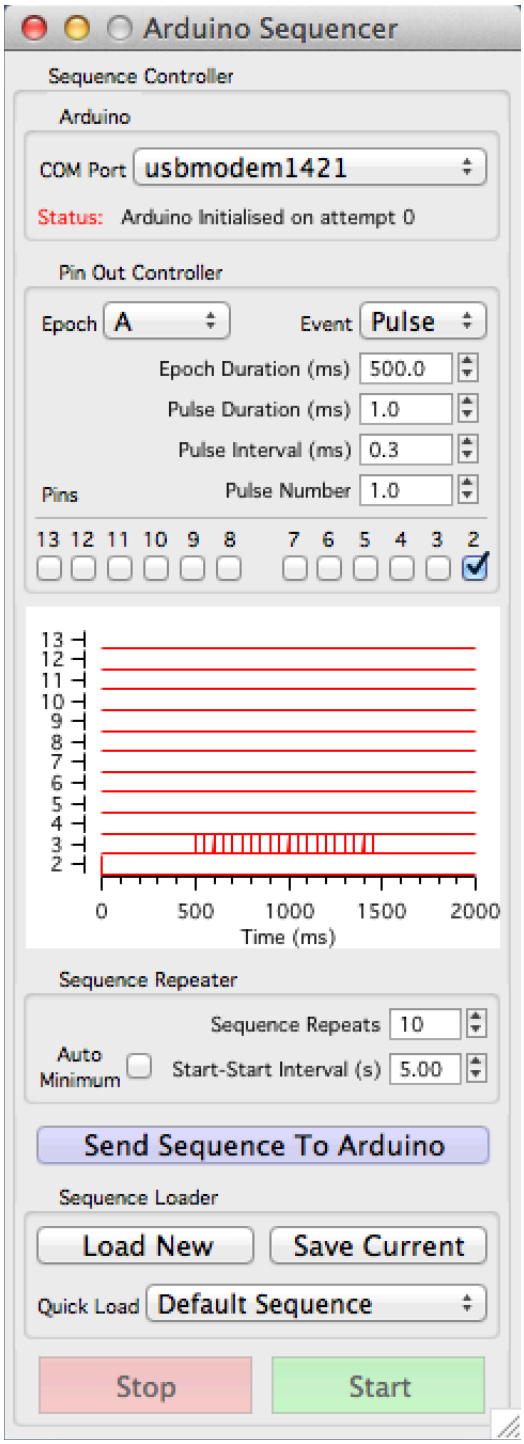
The procedure files for the Igor Sequencer panel needs to be installed in your Igor Pro 6 User Files subfolder.

Create a folder “Arduino” and within that a subfolder “Sequence Files”. This will hold any saved configuration files. When the sequencer starts up, if it can’t find a default sequence file, it will create one but it will fail if there is no “Arduino:Sequence Files” folder present. I have decided not to force Igor to write a new folder itself in case it causes problems.

The following files and folders should be present
Igor Pro 6 User Files:Arduino:Arduino_Sequencer Files:
Igor Pro 6 User Files:Arduino:Arduino_Sequencer_vs1.ipf
Igor Pro 6 User Files:Igor Procedures: Arduino_Menu_and_IncludeFiles.ipf

You will also need to move the VDT2 XOP that comes with Igor Pro into the Extensions folder

When Igor is started, you should now have a menu item called *Arduino* from which you can load the sequencer panel.



Arduino Sequencer Panel
Locating the Arduino

On startup for the first time, Igor may not find the correct device/port. If this happens, then use the popup menu at the top of the panel and try to locate the correct port. It will then attempt to initialise a device on this port. If successful, a status message saying something like *“Arduino initialised on attempt 0”* will appear. I have had some difficulty getting the Arduino to respond consistently to a single command so I have built into the code ways of repeating the instructions so that it will keep trying for a while until it gets the correct response. Therefore, it may initialise on attempt 2 or 3 etc. Once you have correctly located the Arduino, you can overwrite the Default Sequencer file by pressing “Save Current” and next time, the correct COM port should be found automatically. It works very simply and inelegantly by remembering the position in the list of devices returned by VDTGetPorts2 and this may change if you attach other devices to your computer. The information held in “Default Sequence.ibw” will be loaded on start up. If not found, a new file will be written.

Controlling the Arduino

Once the Arduino is recognised, you can set up a sequence. The general principle is similar to that used in PClamp where a series of Epochs (A-H) can be defined. Each Epoch can be defined as being “OFF”, “DC” or “PULSES”. When off, the epoch is not used. In “DC” mode, the Epoch duration can be defined and each pin can be set high (checked) or low (unchecked). In “PULSES” mode, the Epoch duration can be set along with the pulse duration, pulse interval and number of pulses. As more epochs are added, the graph below should show the pulse pattern.

Underneath the graphs are a series of controls that allow you to change the number of sequence repetitions and the interval between them. There is a check box that will allow the interval to be minimised so that the next repeat will start immediately once the first has finished.

When the sequence pattern, repetitions or intervals have been changed or updated, the button “Send Sequence to Arduino” should be enabled. When pressed, this will inform the Arduino of the pattern you want. It takes a few seconds for the information to be sent. The button will then be disabled until another change to the

sequence has been made. Once the sequence has been sent to the Arduino, the start button will become enabled and when this is pressed, the sequence will start immediately. Currently, the "Stop" button does nothing.

Sequences can be saved or loaded using the "load new" and "save current" buttons and a list of sequence files within the Sequence File subfolder can be accessed through the popup menu.

Note on timing and accuracy

Pins 13-8 and 7-2 are addressed simultaneously but each bank is turned on or off sequentially so there will be a small delay (a few microseconds) between the timing of the two different banks of pins. Therefore, if you want simultaneous pulses, it is better to use pins within one bank. The time resolution of the Arduino is reasonable (< 1ms) but not as good as you would get with Nidaq devices. I find this to be a very cost effective solution for triggering electrical stimulation for electrophysiology and for triggering and controlling various imaging devices.

Igor_Sequencer 3.ico sketch

This sketch originated from the web and has been hacked by me. The principle is that the Arduino initialises and then runs an eternal loop, which looks out for commands from Igor. These commands are numbered and can take a series of arguments separated by commas and terminated with a semi colon. Arrays containing values describing the pulse patterns for each epoch are filled up and then run as a sequence when the user requests with the start sequence command.

There are 4 housekeeping commands that are sent from the Arduino to the host computer.

```
kCOMM_ERROR = 000, // Lets Arduino report serial port comm error back to the PC (only works for some comm errors)
kACK = 001, // Arduino acknowledges cmd was received
kARDUINO_READY = 002, // After opening the comm port, send this cmd 02 from PC to check arduino is ready
kERR = 003, // Arduino reports badly formatted cmd, or cmd not recognised
```

Sending "2;" from Igor causes the Arduino to send "Arduino Ready". This is used to confirm contact with the Arduino
Sending "3;", from Igor causes the Arduino to send "Unknown command"

The following 4 functions declared in the sketch are called from Igor Sequencer to control the Arduino:

```
ResetAll_11          // this will loop through all epochs and set values to zero
GetRepsData_12       // This will receive information about repetitions and sequence intervals
GetEpochData_13      // This will receive data about an epoch
StartSequence_14     // This will start the sequence
```

These require the following commands from Igor as a string of numbers separated by commas and terminated with a semi colon.

ResetAll_11 "11,0;"

Resets all of the Epoch values to zero

GetRepsData_12 "12,1,1000,2000;"

The arguments are Command, Repeats, Sequence Duration, Start-Start Repeat Interval with the times in ms.

GetEpochData_13 "13,1,2,500,1,50,10,0,32;"

The above commands would say that epoch1 would be pulse data (a value of 2) 500 ms long, 1 ms duration and 10 pulses at an interval of 50 ms. Port D is pins 2-7 and these will be all off (0). Port B is pins 8-13. A value of 32 means that pin 13 is on (5^{th} bit is on) and all the others are off. The code in the Igor procedure helps explain in more detail. Times are in ms.

StartSequence_14 "14;"

This will start the sequence.