University of Colorado
Anschutz Medical Campus

Optogenetics and Neural Engineering

Arduino Simple 4 Channel TTL Pulse Generator and Controller

Developed and described by Andrew Scallon
Optogenetics and Neural Engineering Core
Abigail Person and Gidon Felsen, Directors
Department of Physiology and Biophysics
School of Medicine, University of Colorado

tl;dr

An inexpensive Arduino controlled, four channel, simultaneous TTL pulse generator and controller with BNC connectors and optional 3D printed housing is detailed. TTL pulse modulation requires programing in the Arduino language, but 2 examples are given such that no prior programing experience is necessary. The first example shows how to set up four independent TTL outputs. The second example expands on this and allows for input into the generator for a closed loop system, controlled by TTL.

Introduction

Research labs often conduct experiments that require precise timing characteristics, synchronized across several signals. Pulse Width Modulation (PWM) is a digital on/off signal that carries information in the relative timing characteristics of the signal. Transistor Transistor Logic (TTL) is a form of PWM where the on state is 5 VDC and the off state is at 0 VDC. TTL is frequently employed to precisely control lab equipment such as lasers. Generation of the TTL signal often comes from expensive lab equipment and is carried through BNC cables from the generator to the equipment. The TTL signal is simple, but must be precisely controlled for correct on/off state and timing considerations. More information on TTL can be found here. Modern microcontrollers can easily accomplish the signal generation of more expensive equipment, for most applications. Arduino is a simple, inexpensive microcontroller that natively outputs TTL signals. The intent of this paper is to outline a method to fabricate an inexpensive TTL signal generator and controller.

Arduino outputs PWM digital signals. When the Arduino is powered correctly, this signal is conveniently at 5VDC, and is therefore TTL. Although the chosen board (the Arduino Uno R3) outputs six channels, only four are employed in this design for ease. If six signals are required, it would be easy to expand on this design. If more than 6 channels are required, it may be possible to connect several Arduinos together, but there may be some delay to consider.

This document will outline the equipment, fabrication, and software required. The equipment is intended to be inexpensive. A model for an optional housing for the Arduino and four BNC connections is offered and maybe 3D printed. Fabrication of the final assembly is simple but does require the use of a soldering iron. The software is free. The Arduino programing language is similar to C+, but it is intended that no previous programing is required to generate TTL signals with this set up. Two example programs are given that are intended to be explained in such a way that signal generation can be designed simply by modifying the code in key places. One example shows how to generate four simultaneous signals, each with a unique pulse pattern that is repeated Ad infinitum. The other example shows how to use

this set up to control a closed loop experiment. The examples can also be used as an introduction to coding and covers if, for, and while statements.

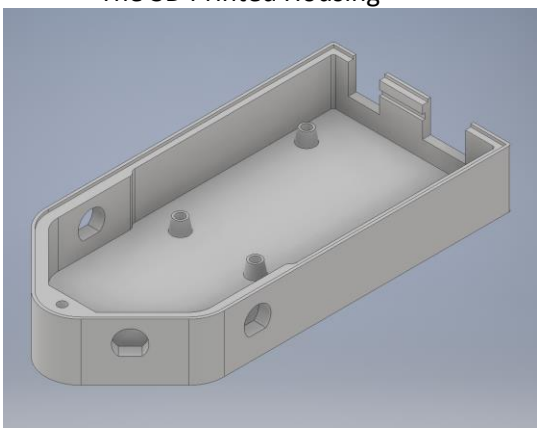
All files are embedded in this document, at the bottom.

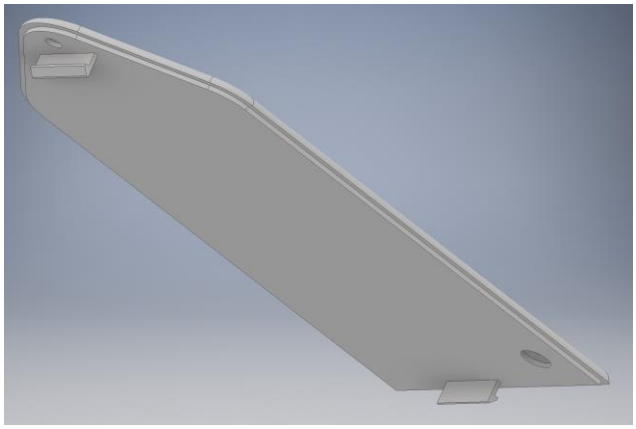
Equipment

Equipment	Supplier	PN	Cost	Unit	Comment
Arduino Uno R3	Adafruit	50	24.95	Each	
Arduino Cable	Adafruit	900	2.95	Each	
Arduino Power Supply	Adafruit	276	7.95	Each	
Female BNC Connectors	Mouser	5227726-1	3.41	Each	Need 4
Hookup Wire 22 AWG	Adafruit	3175	27.5	10x25ft	Need a very small amount
Silicone Caulk	mcmaster	63500334	9.51	Tube	Any caulk will do
Heat Shrink	Amazon	707470898884	10.29	Kit	Only need very small amount
Mineral Oil	Amazon	79567100386	4.59	Bottle	Optional, need very small amount
Solder	Mouser	601-46-502	3.86	Tube	Only need very small amount
3D Printed housing	Varies	NA	40-80		Optional
IR Beam	Adafruit	195	21.67		Optional, example 2
Male BNC Connectors	Mouser	565-4969	8.95		Optional, example 2
Total			165.24		Total
Recommended			109.49		Without general lab parts
Without housing			49.49		No 3D printed housing

The unit is expected to cost $165 total with all parts. It is expected that labs may have some components (like mineral oil or solder), and therefore the cost can be $109. The 3D Printed housing is highly recommended, but varies greatly in price via a supplier (see https://www.3dhubs.com/3dprint).

The 3D Printed Housing

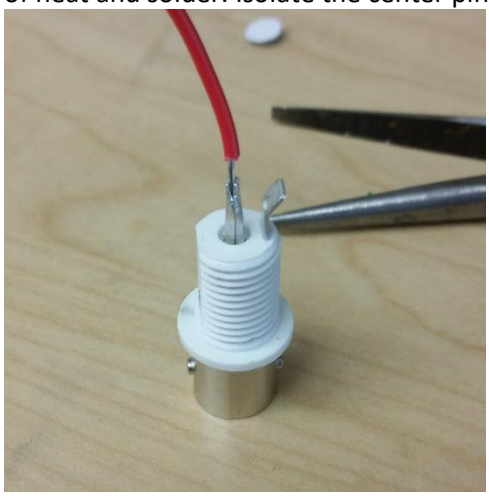




The 3D models are given in the deisgn file format (.ipt) as well as the industry standard for 3D printing (.stl). The parts were successfully printed with nGen at 0.25 mm layer height, 1 mm shell, and 20% infill. The top snaps to the bottom, but a M3 screw can be used as well. A button in the top allows for reset of the Arduino. The Arduino can be powered with USB or a 5 VDC power supply. The BNC connectors are secured on thick walls and are angled to minimize space while maintaining electrical isolation.

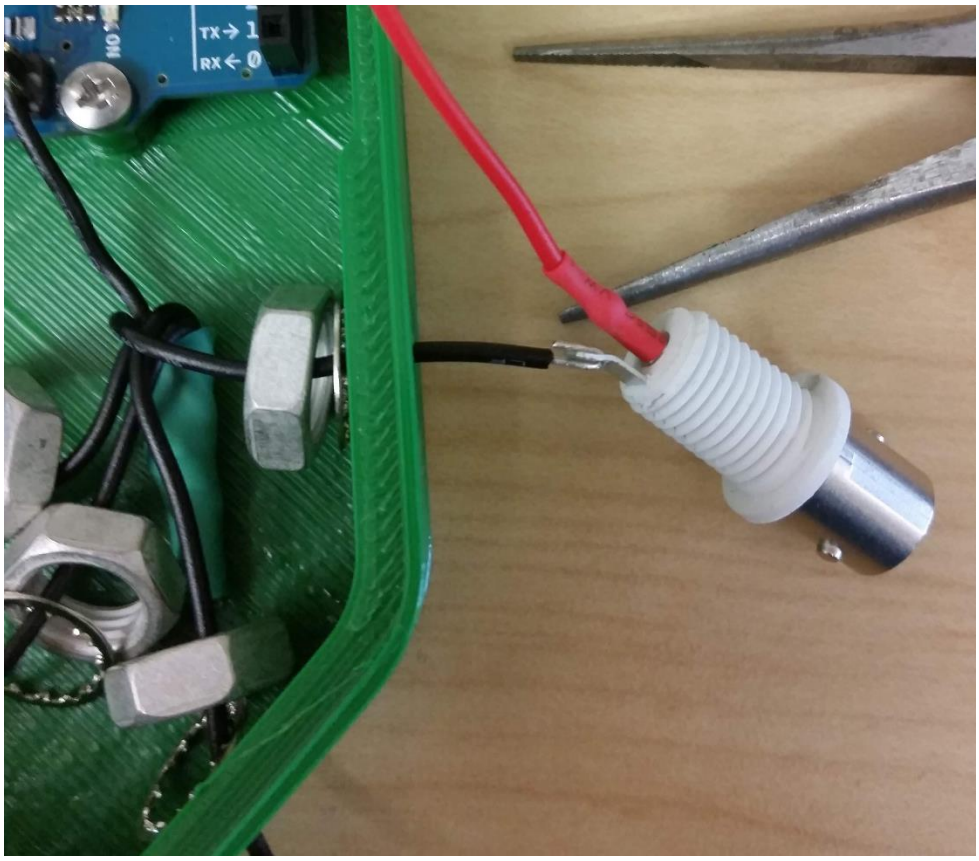
Assembly

1. Secure the Arduino: Use the M3 screws to ‘tap’ the holes prior to placing the Arduino in the housing. Normal machine screws can tap most plastics (ABS, PLA, nGen) easily without the use of taps or self-taping screws. However, the friction may be excessive. An oil (ex: mineral oil) can be used to lessen the friction and displace the friction heat generated during taping, but be sure to clean this out if used. Place the Arduino in the housing and secure with M3 pan head screws. The screws may be metal, but must not touch more than one electrical connection on the Arduino.
2. Create Channel Wires: Cut (4) channel wires of ~8 cm length and strip off ~1 cm from all ends. Insert one end into the center pin of the BNC Connector. This center can be pinched with needle nose pliers to hold the wire until it is soldered. Solder with a generous amount of heat and solder. Isolate the center pin from ground by placing heat shrink over the solder.

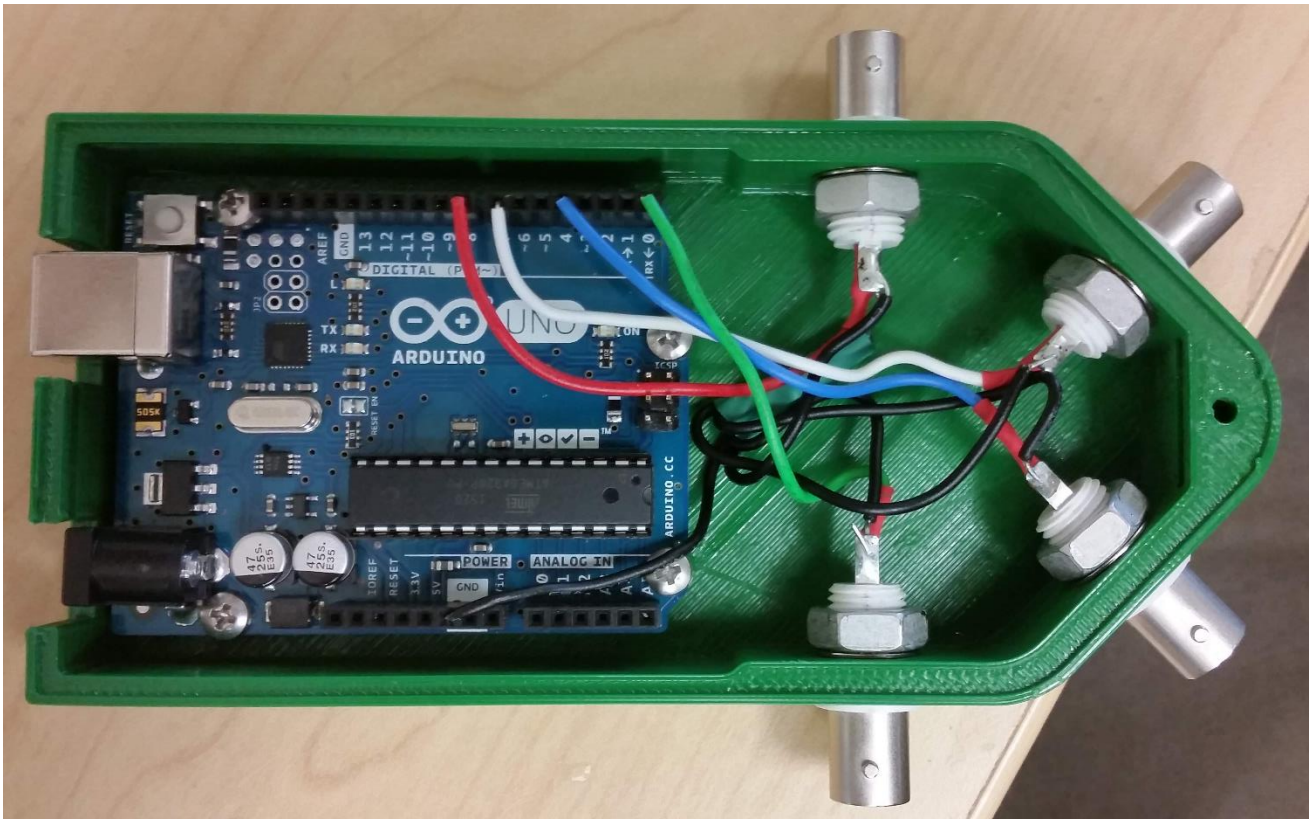


3. Create Ground Wire: Cut (5) ground wires of ~8 cm length and strip off ~2 cm from one end and ~1cm from the other. Twist the longer ended exposed wires together. Solder all together with a generous amount of heat and solder. Insulate this junction (recommend: dip junction in silicone and then heat shrink over this).

4. Connect Ground Wires: Twist all (5) ground wires together at one end and solder together. Connect the one lose ground wire to ground on the Arduino.
5. Place the ground wire junction in the housing below the Arduino. Place the washer and nut onto the wires before inserting the wire through the hole in the housing. Solder the ground wires to the BNC connector outside of the housing (you can insert the wire through the hole in the BNC Connector, fold over, and pinch with pliers for ease). Then insert the channel wire through the hold in the housing and then through the washer and nut. Secure the BNC Connector to the housing by tightening the nut.



6. Connect Channel Wires: Connect each lose wire to the appropriate pin on the Arduino. Channel 1 to pin 2; Channel 2 to pin 4; Channel 3 to pin 7; Channel 4 to pin 8.



7. Snap the housing lid into place and secure with an M3 screw.
8. It is a good idea to label the TTL number (as well as Pin numbers).

Programing and software

1. The Arduino is programed through a free software (Arduino IDE) that can be found here. Download, install, and open per your OS. Connect the Arduino via the USB Cable. In the

Arduino IDE, select Tools > Boards > Arduino Genuiono Uno. Then select Tools > Port > then select the board connected to the USB (typically COM4)

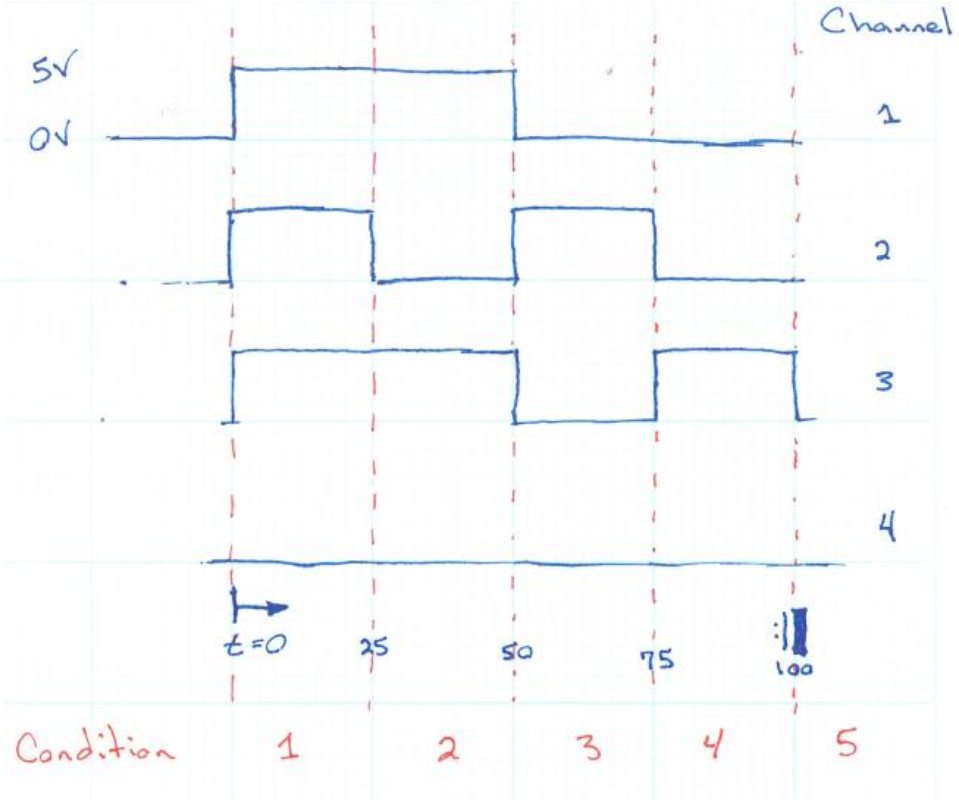
2. Open the desired file in the Arduino IDE via File > Open. Upload this to the Arduino by selecting the Upload button (right arrow button in the top left). A progress bar is indicated at the bottom right. Wait for the Done Uploading to appear. The Arduino will automatically restart and will immediately start outputting the TTL pulses.

Example 1: Simple 4 Channel TTL Pulse Generator

This example will describe how to generate four simultaneous signals, each with a unique pulse pattern that is repeated Ad infinitum. If fewer than 4 channels are required, you can simply leave the other channels as off (LOW), or they may be commented out (place "//" in the beginning of a line).

Hardware set up: Connect all four BNC's to a four probe Oscilloscope. All channels on the oscilloscope should be set to 5 V divisions, 100 ms scales, and trigger off of channel 1.

To start, **trace out the desired pulsing on paper**, with each trace on the same timeline. In this example, the first channel will output a 50 ms pulse, then remain off for 50 ms. The second channel will pulse on and off for 25 ms each. The third channel is to be off only during the second on pulse of channel 2. The forth channel is to remain off the entire time. This would look like:

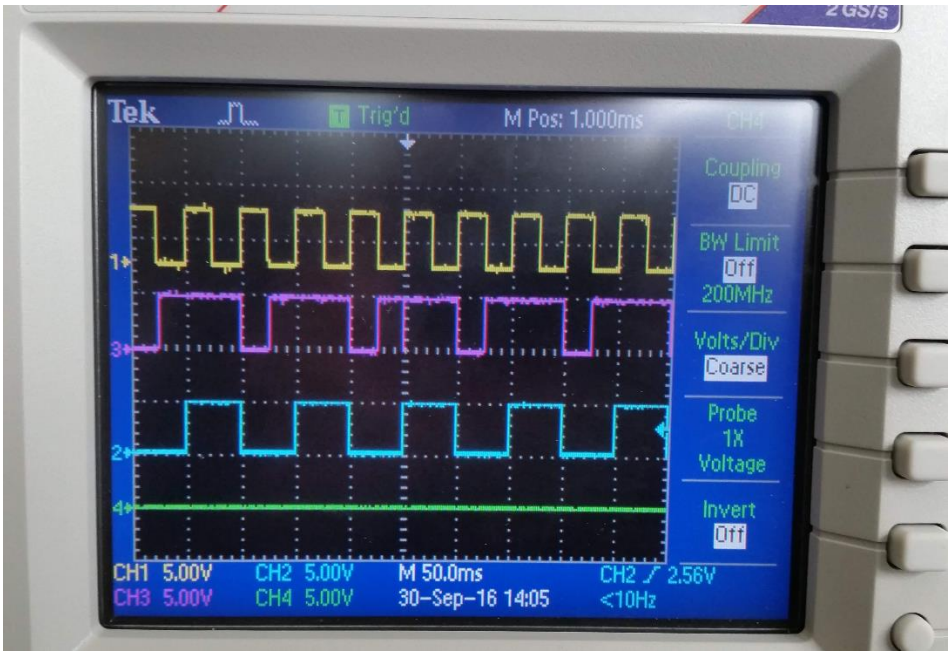


The Arduino will act like a person at a light switch, turning on or off the channels when directed to do so, and simply waiting in-between. **Any time any of the switches changes, the 'condition' of the Arduino changes**. We can therefore define the condition of the Arduino across the entire pulse train. For the given example, we can define 4 or 5 conditions (more on this later). Each condition in this chosen example lasts 25 ms, but the durations of conditions can be independent. **Any time any of the TTL's change, draw a vertical line and label it as a new condition. We can then define the state of each TTL channel for a condition. We can also define the time (t) for the duration of each condition.**

Open the Arduino_Simple_TTL.ino file with the Arduino IDE. It is broken into sections. The first section is an introduction. The next section defines what pins we will be connecting to on the Arduino. We then run a set up to tell the Arduino that these are all output digital pins. These first sections will typically remain unchanged. The next section begins to deal with the desired pulses. Here we define the duration of each condition. The given example has four conditions last 25 ms. You could also consider a fifth condition of zero time, but wherein all pulses are turned off (more on this later).

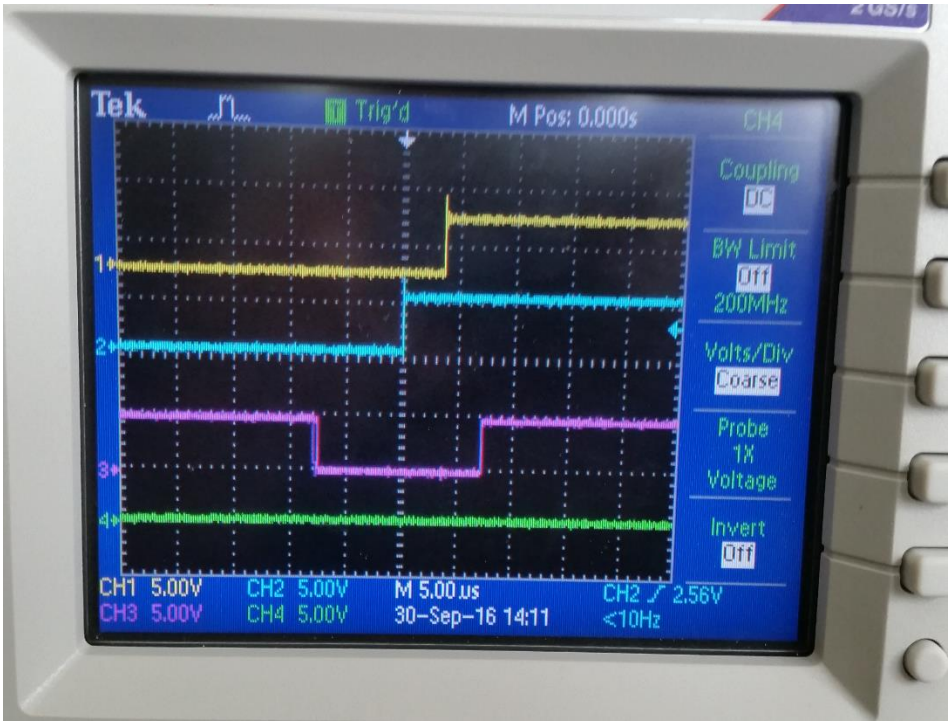
The final section is the heart of the code. This is the looping code that the Arduino will loop/repeat through. We label each condition, tell the Arduino how to set the switch of each channel,

and then tell the Arduino how long to wait in that condition. Uploading this to the Arduino will output the following to an oscilloscope:



(Note that channels 2 and 3, although appropriately labeled by color, are swapped in horizontal placement in the picture.) The outputs are well synced, accurately timed, and repeat continuously. However, looking at channel 3, you will note that most waves appear uniform, but the middle pulse has a divot in the middle. If one were to zoom in, you would see that all pulses on this channel has this feature. This is because, although we chose condition 5 to be of zero ms, we still told the Arduino to switch off all pulses, including channel 3. The Arduino turns channel 3 off and then on between the pulse trains. It is therefore recommended that the state between pulse trains not be considered as a condition, but you must at least consider how the pulse train end matches the beginning.

Timing considerations

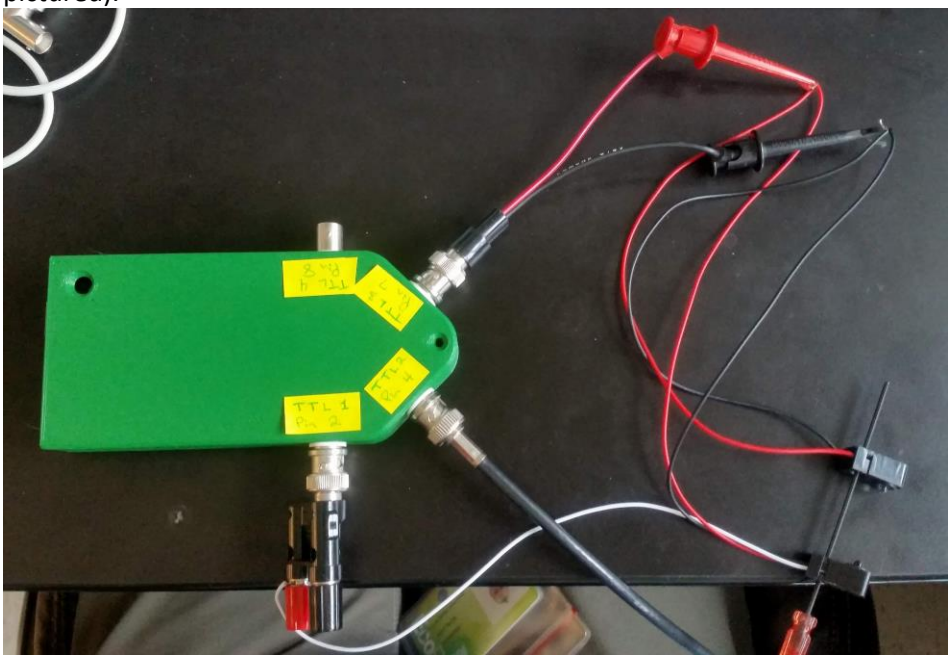


When you zoom in on the oscilloscope trace, you will note that there is a delay in when the channels are turned on; they are not turned on all at once. This is a limit of the Arduino clock speed. For the Arduino Uno. There will be at minimum a 4 us delay between turning on two channels. If 6 channels are used, this could result in a delay of 24 us between channels. For most biological applications, this is not an issue, but it is strongly recommended that this be considered for any experiment. Note that the Arduino can be over-clocked (sped up at the risk of damaging parts). The Arduino Duo has a clock speed 5.25 faster, and would presumably decrease this delay by a factor of 5.25. The order of which channels are turned on is also tunable, but all this is beyond the scope of this paper.

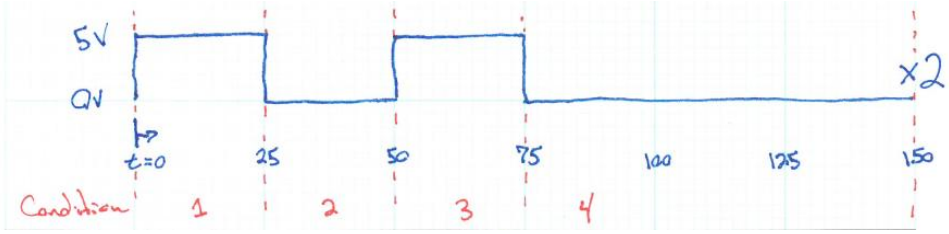
Example 2: Arduino Simple Closed Loop

This example will show how to use the Arduino to deliver a TTL pulse only after some TTL input is received. As an example, suppose we wish to set up an experiment wherein an animal is trained to poke its nose in a port. We wish to sense when this occurs and then deliver an optogenetic stimulation; a train of pulses. We can detect the nose poke with a very inexpensive IR Beam sensor. In this sensor system, an LED emits IR light (not detectable by human nor mouse vision) and a sensor senses the light. When the IR beam is broken by the mouse nose, we can detect it and deliver this information to the Arduino via a TTL signal. We can then use another channel to output a TTL signal to a laser to optically stimulate a brain region of interest. A third channel will be set to continuously on to power the IR light and IR sensor.

Hardware setup: Connect TTL2 (the optogenetic output) to an Oscilloscope set to 5 V divisions, 100 ms. The IR LED and sensor can be powered through the same pins in parallel. It may be handy to solder the like colors together (that is, solder the IR LED red wire to the IR sensor red wire). Connect these wires to a male BNC on TTL3 (power for the LED and sensor). Finally, connect the white wire on the IR sensor to the positive terminal of TTL1 (there is no ground for this sensor; the Arduino will use its own ground). The IR LED and sensor can be kept aligned to each other with some rod (Allen wrench pictured).



Again we should **begin by defining and drawing the TTL pulse train we would like to use**. For this example, we would like to deliver two evenly spaced pulses with a long delay at the end. We want this pulse train delivered twice. It is important to note that this pulse train be delivered only twice per nose poke.



Open the Simple_Closed_Loop.ino file in the Arduino IDE. Again there is an introduction. The next section defines the pin output on the Arduino. Although we have 4 pins physically mounted, we are

only using 1-3, so the rest are commented out ("//"). We next define variables that Arduino will use to monitor the state of the system. These are internal numbers that Arduino uses. The next section, the condition section, where we define the timing characteristics of the optogenetic stimulation. This is the waveform drawn above (see Example 1 for more information).

The next section tells the Arduino to treat the pins as either output or input. We assume that the initial state of the sensor will be that the beam is on and detected. On is high. We therefore set the initial state of the sensor as high. We next set the initial state of the optogenetic TTL2 to low, as we assume the optogenetic stimulation should start as off. Finally, we set TTL3 as high. The remaining code does not change this value, so as long as the Arduino receives power, this channel will remain at 5 VDC. We can use this to power *low current* components, such as the IR LED and sensor. Components that draw more than 40 mA (or shorting this circuit) can damage the Arduino.

The next the section is the code that Arduino will continuously loop through. It starts by reading the state of the sensor. It then checks if state of the sensor is low (IR Beam broken). This 'if' command can be read as: if the sensor state is low, then execute the indented code, if the sensor state is not low, do not execute the indented code, move to the 'else' command. The else section gives opportunity to have the Arduino do something while the animal is not in the nose poke.

If the sensor is low (the beam is broken), the 'if' command is true and the Arduino will look at the indented code. The 'for' code can be read as: for as long as variable (i), originally set to zero, is less than variable numrepeat (here 2), execute the further indented code, but then at the end add one to the variable (i++). So, for the first time Arduino sees this code, i=0, so it executes the further indented code, adds one to i, and starts again. This time, i=1, which is less than 2, so it again executes the further indented code and adds one to 1. Now i=2, which is not less than 2, so it does not execute the condition code.

The further indented code, or 'condition' code (which outputs the pulse train) can be understood by working through Example 1: Simple 4 Channel TTL Pulse Generator.

We then say that if we have run through the optogenetic stimulation numrepeat times, reset the sensor to high (this was the same as what was done in the initialization).

If the code were to end here, the Arduino would start the loop over. That is, immediately after delivering the optogenetic stimulation, it would start over and read the sensor. The stimulation delivered is very short (2*150 = 300 ms) and it is probable that the mouse could still be in the nose poke. The sensor would be read as low, and another optogenetic stimulation would be delivered. But we said in the beginning we only want one optogenetic stimulation per nose poke. We should therefore wait until the mouse leaves the nose poke. A while command can achieve this. The while statement can be read as: while the sensor state is low (off), delay 500 ms (.5 seconds), read the sensor state again, repeat.

Supplementary Files

Files may be opened directly from this Word Document, or maybe copied into a folder of your choice.

Screenshots of Downloading and Pro

Screenshots of installing Arduino Software and uploading programs to the Arduino.

Arduino_Simple_TTL.ino

The Arduino code for Example 1: Simple 4 Channel TTL Pulse Generator

Arduino_Simple_Closed_Loop.ino

The Arduino code for Example 2: Arduino Simple Closed Loop

Arduino simple 4 channel ttl bottom.ipt

The Design File (for modification) of the 3D housing bottom

Arduino simple 4 channel ttl top.ipt

The Design File (for modification) of the 3D housing top

Arduino simple 4 channel ttl bottom.stl

The 3D Model for printing the housing bottom

Arduino simple 4 channel ttl top.stl

The 3D Model for printing the housing top

Please acknowledge our facility in your publications. An appropriate wording would be:

"Engineering support was provided by the Optogenetics and Neural Engineering Core at the University of Colorado Anschutz Medical Campus, funded in part by the *National Institute for Neurological Disorders and Stroke* of the National Institutes of Health under award number P30NS048154."