Fiach O'Donnell, R00082543

Hugh McCarthy

Dissertation (MUSC9004)

19 August 2019

An audio **C**ontrol **S**urface built using the **A**rduino platform and **W**ireless capabilities

CSAW

Fiach O'Donnell

R00082543

THIS THESIS IS PRESENTED AS PART OF THE REQUIREMENTS FOR THE MSC. IN MUSIC & TECHNOLOGY, AWARDED BY CORK SCHOOL OF MUSIC

Supervisor: Hugh McCarthy

Submitted to Cork Institute of Technology, August 2019

Declaration of Originality

- I declare that I am the sole author of the written document here enclosed and it is my own original work, except where otherwise accredited. All sources have been cited and acknowledged.

- I certify that I have compiled the accompanying thesis in accordance with Cork Institute of Technology's student regulations on plagiarism, which I have read and understood.

- I state that this document has not been presented or submitted for award at any other institution.

Candidate Signature: _____ Date: _____

Supervisor Signature: _____ Date: _____

Acknowledgements

Many thanks to Hugh McCarthy, Paddy Collins and Keith Clancy for their assistance and advice throughout the year. Also, a big thank you to Damien O'Brien for coming through with the finished desk surface, my course colleagues for helping me get through the year and my parents; for immeasurable guidance and letting me hijack the family room for my own personal workshop the past 2 months.

Abstract

Audio control surfaces are practical tools for editing and mixing in a digital audio workstation. Despite recent technological advancements, however, there is a lack of robust physical commercial controllers with seamless wireless functionality.

This paper describes the design and implementation of CSAW – an audio **C**ontrol **S**urface built using the **A**rduino platform and **W**ireless capabilities. CSAW aims to remedy limitations of mobile control applications and certain hardware controllers through devising a proof of concept control surface and software platform using sophisticated open-source applications and wireless communication protocols.

Abbreviations

ADC	Analog-to-Digital Converter
CAD	Computer Aided Design
COM	Communications port
CSAW	Control Surface using Arduino Wireless capabilities
DAW	Digital Audio Workstation
DC	Direct Current
EQ	Equalization
ESP	ESP8266 WiFi module (ESP-01)
FX	Effects plugin
GND	Ground connection
GPIO	General-Purpose Input/Output
HUI	Human User Interface
IDE	Integrated Development Environment
INH	Inhibit (enable) control pin
IoT	Internet of Things
LED	Light-Emitting Diode
MCU	Mackie Control Universal
MIDI	Musical Instrument Digital Interface
MQTT	Message Queuing Telemetry Transport
OSC	Open Sound Control
PCB	Printed Circuit Board
Pd	Pure Data
SDLC	Software Development Life Cycle
SQL	Structured Query Language
TCP/IP	Transmission Control Protocol/Internet Protocol
TTL	Transistor–Transistor logic
USB	Universal Serial Bus
VCC	Voltage at the Common Collector
VDD	Voltage Drain Drain
VEE	Voltage at Common Emitter
VPL	Visual Programming Language
VSS	Virtual Switching System

Contents

Table of Figures

1) Introduction

1.1 Introduction

The advent of smart phone technology and software applications in the modern age has undoubtably seen an increased interest in wireless and remote audio control within the audio production spectrum. The consolidation of music with the Internet of Things (IoT) in recent years has surpassed the expectations of engineers and enthusiasts alike, who continue to discover and implement new means of audio manipulation at a rapid rate (Turchet, Fischione, Essl, Keller, & Barthet, 2018). Through these technologies and common commercial peripherals, this is now very easily achievable and easier again to develop upon as future demands dictate.

Despite these advancements, however, there is still quite a lack of robust wireless capabilities for physical controllers/control surfaces on the market. Big name brands such as Behringer influence the pack with products like the X-Touch, but even so, this type of control is not as widely implemented as expected, considering the universal adoption of WiFi in the late 90s (Behringer, 2014). Understandably, companies will always cater to as broad of an audience as possible, and due to the widespread acceptance of tablets and smartphones in the average household, this trend was not unanticipated.

Nevertheless, for audio professionals, the necessity of a physical controller is imperative, given the frequent need to make multiple changes simultaneously and more accurately, in contrast to the visual limitations and un-natural feel of changing parameters on the small screen. Combined with wireless facilities, this can provide an additional layer of flexibility within the workspace, while maintaining the same degree of mobility and comfort.

The **CSAW** (**C**ontrol **S**urface using **A**rduino **W**ireless capabilities) project aims to remedy this through utilising and consolidating existing technologies and hardware to construct a functional wireless physical control surface for audio manipulation. Through the use of sophisticated applications and communication protocols, along with a fundamental electronics environment with elemental components, a proof of concept controller and software platform will be devised to enable the audio engineer to perform mixing and recording functions in an easier, more efficient manner than its software counterparts.

1.2 Motivation & Problem Statement

The progress of single-board computers and microcontrollers has resulted in a wide variety of projects and solutions in the audio domain. CSAW aims to continue to build on these developments by employing Arduino and fundamental electronics to provide a practical, stand-alone solution, capable of performing wireless functions that are missing in many modern, low-budget commercial controllers. The project was devised to create a controller that would introduce an analog aspect - still performing the same level of digital processing but filling the absent role of wireless physical control surfaces, which have largely been replaced with singular mobile software applications despite being less user-friendly and providing lower level of accuracy.

1.3 Objectives of the Project

The primary objectives of the CSAW project are:

- to construct a proof of concept audio control surface with wireless capabilities that is proficient to work effectively with a number of digital audio workstations.
- to develop an ergonomic, user-friendly solution that could match the functionality of well-known commercial controller products.
- to examine the competency of microcontrollers with basic electronics theory in the domain of music technology, alongside existing open-source technologies/solutions.

1.4 Chapter Outline

Section 1 introduces the CSAW project, its main objectives and its feasibility as a proof of concept within the audio control surface domain. In Section 2, the state of the art and some commercial solutions are examined, along with pertinent technologies and background. Section 3 outlines the numerous aspects of CSAW's design and development processes. Section 4 describes the implementation and construction of the project itself based on these designs, with some testing methodologies. Finally, Section 5 examines the scope of the project in the future through potential developments and adjustments to the existing solution.

2) Literature Review & State of the Art

2.1 Control Surface

An audio control surface refers to a device or interface that "allows the user to control a digital audio workstation or other digital audio application" (wikipedia.org, 2019). Quite often, the phrases "control surfaces" and "controllers" are used interchangeably, though the former is more specific for DAW control (hence the adoption of the phrase "DAW controller") and often resembles an analog mixing console with faders and various knobs/buttons. These range from the primitive, such as Behringer's X-Touch Mini (a considerably stripped-down version of the aforementioned X-Touch) with its single fader and 8 rotary potentiometers for track volume control, to the more elaborate, like Avid's Pro Tools S3, which allows for a wider range of functionality such as automation, routing management, etc. (Avid, 2014)

CSAW will act like such an interface, allowing for basic DAW and track control implemented through Arduino and WiFi capabilities. The choice to make a mobile physical control surface was based on recent trends which appear to have shifted focus primarily on software apps, such as DAW Remote HD (DAW Remote HD, 2019). These applications do not provide the same level of efficiency or accuracy as physical controls. While the CSAW controller will not be mobile in the same sense of fitting in a jeans pocket, it will still provide a level of mobility in the production space.

At its core, CSAW's design will be closer to a combination of control surface and analog console than to a typical control surface, based on its parameters. Controller layout is closer to a basic analog 12-channel mixer with some extra functions than any typical control surface. Most common DAW controllers do not include specific EQ and compression functions, instead including buttons & knobs that are programmable to DAW functions.

Since this project utilises existing technologies instead of developing a project-specific plugin or executable, this programmability is still achievable albeit more difficult without knowledge of the underlying code and patches. For this reason, the CSAW controller will include analog sensors with pre-mapped controls for the more common DAW functions.

2.1.1 Commercial Controllers & Analog Mixers

Before undertaking any design or development activities, it was important to consider and research commercial control surfaces and analog mixers, noting what areas these products succeed at as well as recording their faults. These notes would influence the design and functionality of the CSAW controller over the course of the project.

Behringer X-Touch

The Behringer X-Touch has 8 assignable rotary controls along with 9 motorized faders and many different buttons for key DAW functions (Behringer, X-TOUCH: Quick Start Guide, 2014). It provides universal control, understandable by most DAWs via the Mackie Control Universal (MCU) protocol, allowing for easy hardware interfacing to a DAW via MIDI (soundonsound.com, 2019).

The X-Touch also has WiFi remote control functionality, achieved through directly connecting the device to a router, configuring its network settings, creating a new network session on the source computer and using Mackie Controller and drivers like rtpMIDI (for Windows) for communication with a DAW (rtpMIDI, 2019).

Figure 1. *Behringer X-Touch (thomann.de, 2019)*

The X-Touch requires a lot of setup and configuration for WiFi functionality. Making use of DAW built-in OSC/MIDI capabilities instead would require only creating a single software

control surface without any additional drivers or applications. Ideally, a physical control surface should also have a WiFi transceiver for easier connectivity to a pre-existing network instead of over Ethernet. CSAW aims to integrate both of these.

Avid Pro Tools S3

A streamlined version of the Pro Tools S6 control surface, the S3 has 16 channel strips with motorized faders and 32 rotary encoders that also act as pushbuttons, all housed in a compact design for small production spaces (Avid, 2014). One of the more up-market control surfaces on the market, the S3 utilises EuCon, an object-oriented protocol for connecting control surfaces to applications. As the name suggests, the S3 is primarily geared towards Pro Tools though can work proficiently with other DAWs through this protocol.

Figure 2. *Avid Pro Tools S3 (thomann.de, 2019)*

Issues with many of these more sophisticated controllers often come with price and overabundance of functions. At its high price range, the S3 has more functionality than more affordable control surfaces but much of this could be considered excessive. Control surfaces are an accessory, albeit an important one, and at the S3's price a high-quality digital mixer could be purchased instead, doubling as a controller, audio interface and used for live sound

mixing. It is important to strike the right balance between functional and cost-efficiency to optimise audio production.

The CSAW project will make use of simple, affordable components while keeping to a compact design with the core functionality expected of a control surface. Since one bank of 8 channels is proposed, no motorized faders will be integrated (much of the high cost of control surfaces can be pinned on these alone). However, this could be extended upon in the future. A full budget and list of materials can be found in Appendix C.3.

Behringer Xenyx X1222USB

By its nature, the X1222USB is not directly comparable to the other controllers specified in this section (Behringer, Xenyx X1222USB User Manual, 2019). Design-wise, mixers and control surfaces are closely related, though mixers provide some obvious additional functionalities. The X1222 has an intuitive layout and compact size, greatly influencing the design layout for the CSAW controller, with some minor differences [Fig.3].

Figure 3. *Behringer Xenyx X1222USB (sweetwater.com, 2019)*

For example, the CSAW controller will not house a monitor send bus, instead opting for a single auxiliary output - also to be used as an FX send (reverb by default). All 8 channels being controlled act as mono channels instead of the 4 mono and 4 stereo channel strips on the Xenyx. Faders used in the project are dual analog potentiometers, which could be built upon to control left/right channels in the future.

2.1.2 Similar Projects & Software

The concept of implementing Arduino for the creation of audio controllers has been touched upon in the past. Previous hardware projects have generally centered around instrument-based controllers, over analog-style DAW controllers, and almost all universally utilise MIDI instead of technologies like OSC for wider compliance. Section 2.3.3 outlines benefits of utilising OSC.

Hardware Projects

A host of MIDI controllers and instruments using Arduino exist in the public domain. A quick-start article from MakeUseOf.com printed in July 2016 outlines the process of building a very basic controller using Arduino and pushbuttons in only 7 minutes, displaying how easy this process can be with the right components (makeuseof.com, 2019). The same site also produced a list of public Arduino controller projects in 2018, consolidating button controllers, drum machines, footswitches and more (makeuseof.com, Arduino MIDI Controllers, 2019). Clearly, Arduino has already proven its worth in the public eye as a reliable platform for audio manipulation.

In particular, Victor Frost's "OpenTransport" project hosted on the Hackaday hosting platform is particularly relevant, an "Arduino-Based Open-Source Transport Controller for DAWs and DVWs" (hackaday.io, 2019). The project was developed with the ambition of building an external (i.e. physical) control surface for less than the cost of a professional model. In a similar vein, several GitHub repositories online also aim to provide control surface capabilities to Arduino, a more cost-effective solution (tttapa/Control-Surface, 2019). CSAW is a little more ambitious with its design but ultimately intends to achieve this same goal.

Software Applications

Software projects generally work as standalone applications instead of utilising Arduino or other platforms. Touted as “the original touchscreen MIDI and OSC control app”, TouchOSC is a fully modular software control surface, capable of sending OSC and MIDI messages over WiFi via an iOS or Android application (hexler.net, 2019). While this is more extensible by design, the likes of TouchOSC, DAW Remote HD and similar apps do not provide the same level of robustness and accuracy as physical, analog-style controls. Yet again the debate of analog vs. digital control continues.

2.2 Hardware & Electronics

2.2.1 Arduino

Arduino is an open-source hardware and software platform for building electronics projects (arduino.cc, 2019). The company manufactures programmable circuit boards, often referred to as “microcontrollers”, for constructing interactive objects or environments which are programmed through the Arduino programming language and its integrated development environment (IDE). Arduino’s flagship microcontroller is the Uno, though all boards typically follow the same general layout of microprocessors, controllers, pins (analog, digital I/O, serial) and more, with some variations.

CSAW employs the Arduino Mega 2560 (Rev3) for sensor input - geared towards more complex projects [Fig.4]. It can be viewed as the controller’s compiler, collecting data from each sensor and forwarding it to an application. The Mega was chosen over other microcontroller boards due to its specifications, specifically larger memory space and additional analog and digital I/O pins (Appendix B.1).

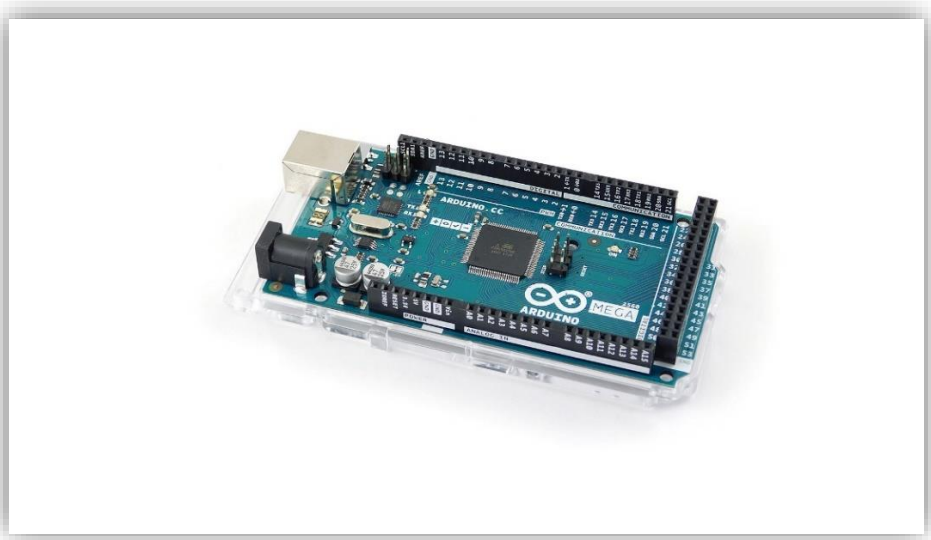


Figure 4. *Arduino Mega 2560 Rev3 (nettigo.eu, 2019)*

2.2.2 ESP8266

The ESP8266 is a microchip module that provides WiFi capabilities to microcontrollers via an integrated TCP/IP protocol stack and given an available network (espressif.com, 2019). It allows for the extension of applications to the world of IoT and can provide network access from Arduino to the outside world. The ESP8266 is also capable of acting as a web server or host and even perform basic microcontroller functions, depending on model.

The CSAW project makes use of the ESP-01 module by Ai-Thinker allowing for serial communication between Arduino and the module by way of RX/TX pin connections (AI-Thinker, 2015) [Fig.5]. ESP-01 was chosen based on its compact size, flexibility, low-cost and active user community. Unlike most of the other components used in this project which facilitate both 5V and 3.3V input, the ESP-01 exclusively requires 3.3V to operate properly (specifications in Appendix B.2).

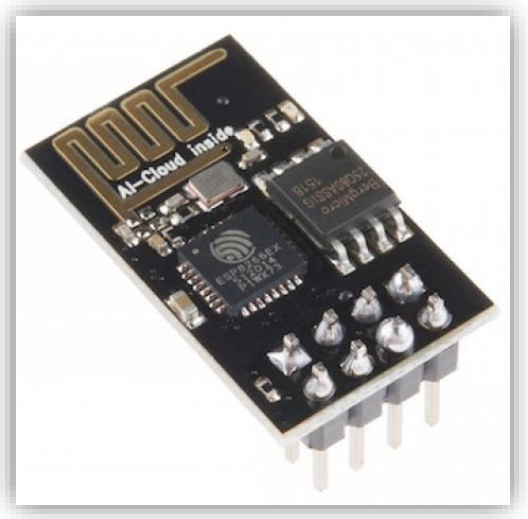


Figure 5. *ESP8266 ESP-01 WiFi Module (potentiallabs.com, 2019)*

2.2.3 Potentiometer

Potentiometers (or “pots”) are three-terminal resistor devices with rotating or sliding contacts connected to a voltage source. They act as both variable resistors and voltage dividers, where rotating or moving the contact in a specific direction either increases or decreases the pot’s resistance, respectively increasing or decreasing the fraction of its electric potential/output

voltage. Two types of potentiometer are used for this project: sliding potentiometers (acting as volume faders) and rotary potentiometers (knobs – everything else) [Fig.6].

All potentiometers used in the project are 10kΩ linear taper pots, i.e. they have an accurate relationship between their slider position and the resistance at that position. Normally, logarithmic potentiometers are used for audio applications, power amplifiers etc. due to human hearing responses to logarithmic changes in sound levels (resistorguide.com, 2019). This would be the case if the signal was fed through the potentiometer directly to make changes in sound volume sound natural. For the purposes of mapping for digital volume control and other functions in applications like Pure Data (Pd) and DAWs, linear potentiometers are better suited.

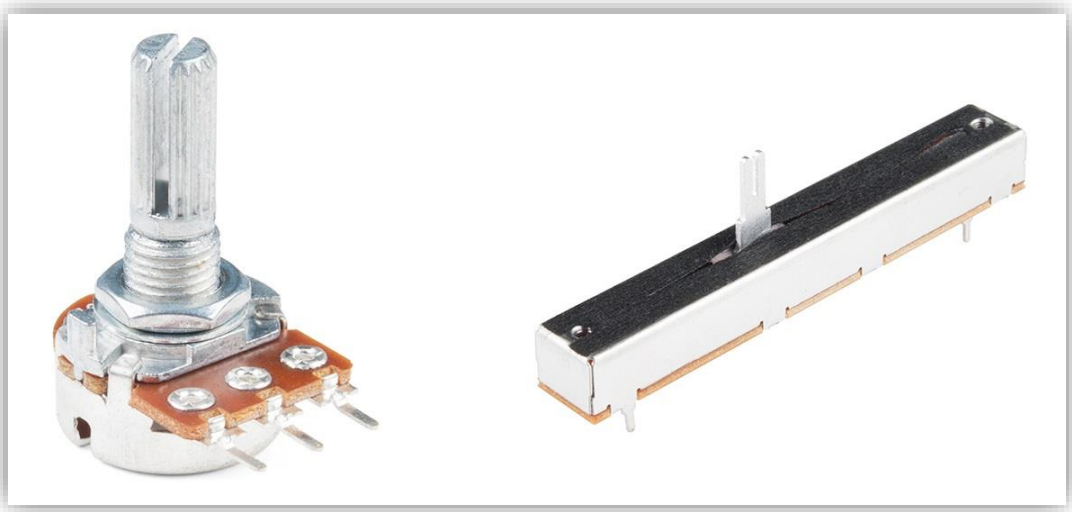


Figure 6. Rotary (left) and Slide (right) potentiometers (sparkfun.com, 2019)

2.2.4 Other Circuitry

Pushbuttons are simply button switches that control a specific action or process. For the purposes of this project, they are used for channel soloing and playback/recording functions.

CSAW also makes use of analog multiplexers to extend the number of analog inputs available to the Arduino Mega. These devices are implemented and examined further in Section 4.1.2.

2.3 Software & Proposed Technologies

2.3.1 DAW

In its most basic form, a Digital Audio Workstation is an application software used for the recording, production and manipulating of audio. Occasionally the term "DAW" is used to refer to a greater element, i.e. "a PC or Macintosh equipped with sound cards and software for editing and processing digital audio", but more commonly it describes the software itself (Leider, 2004). Regardless, DAWs always have some form of interface allowing for basic functionality like mixing track volumes, playing/stopping & recording audio from inputs, applying effects and filters to audio etc.

CSAW's proposed architecture requires a DAW with Open Sound Control (OSC) support. A number of these exist such as Logic Pro, Ardour etc. and some example templates and config files to get these applications working with CSAW are available in the project's GitHub repository (odonnellf/CSAW, 2019). However, for primary testing the REAPER production software was used for audio manipulation (reaper.fm, 2019).

REAPER

REAPER was selected for principal testing purposes over other DAWs for a number of reasons, specifically:

- multi-platform support across all Operating Systems, with an easy-to-use interface.
- large flexibility and adaptability for externals & peripherals
- considerably active and helpful userbase, constantly working on integrating REAPER with other projects.

Figure 7. REAPER logo (reaper.fm, 2019)

Custom actions in REAPER were constructed with mappings from sensors connected to the embedded Arduino microcontroller. These actions are recognised via OSC once the middleware software (Pure Data), translates and scales them into a numerical format understandable by REAPER. This logic is applied to a number of functions, along with FX plugins, through separate OSC messages. Section 4 details this implementation process.

2.3.2 Pure Data

Developed in 1996 to remedy deficiencies in the Max visual programming language (VPL), Pure Data is an open-source VPL for creating multimedia applications and is a member of the "patcher" programming family (Puckette, 2016). Primarily geared towards interactive computer music and audio processing, Pd is based on algorithmic functions (known as "objects") and data flow between these objects through visual connections ("patch cords"). It can be downloaded in several distributions and runs on a large variety of embedded/non-embedded systems.

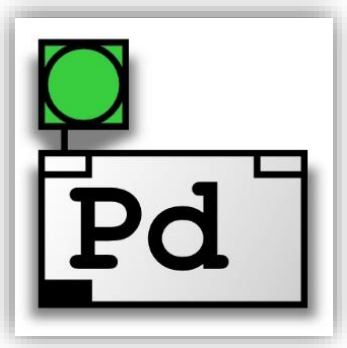


Figure 8. *Pure Data logo (minchee.org)*

Pd will be utilised as middleware software between the CSAW control surface and a DAW. In this context, Pd acts as both event listener and handler; a) listening for messages transmitted via the controller's Arduino and WiFi functions, b) translating/processing and scaling this data so it is understandable by a DAW/REAPER and c) connecting and broadcasting this data to REAPER via the OSC protocol. While the embedded Arduino software will collect and transmit voltage patterns related to individual potentiometers and buttons, Pd will be responsible for processing the data so that REAPER can understand it and commit the appropriate software changes.

CSAW utilises the Pd-l2ork “Purr Data” distribution, a fork of the now discontinued Pd-Extended, to avail of additional libraries, and is available on GitHub (agraef/purr-data, 2019).

2.3.3 Open Sound Control

Open Sound Control is an open, message-based protocol for communication among computers, sound synthesizers and other multimedia devices and applications (Wright, Freed, & CNMAT, 1997). Originally developed at the Center for New Music and Audio Technology at UC Berkeley (CNMAT), OSC is often used in the field of new interfaces for musical expression and, in particular, is optimized for network distributed music systems. OSC is recognisable across a large host of platforms and uses its own customisable URL-style naming scheme for dynamic messaging (opensoundcontrol.org, 2019),

OSC was selected for CSAW over MIDI and other communications protocols for several reasons. Though MIDI is widely recognised as the technical standard and supports many applications, OSC’s user-defined message structure and robust ability to define different data types made it an obvious choice for a project of this calibre, which required a protocol catered towards networking technologies, explicitly WiFi (Freed, Schmeder, & Zbyszynski, 2007). Its human-readable messages also greatly facilitated processing and troubleshooting operations.

Figure 9. *OSC vs MIDI/XML message formats (jordansperfectech.blogspot.com/2012/04/)*

2.4 Analog & Digital Control

Like most sophisticated microcontrollers, the Arduino Mega detects two principal kinds of signal based on its pin types; digital and analog (learn.sparkfun.com, 2019). Digital signals have a finite set of values – normally two. These values are binary in practice, typically representing sensors/devices that have two states: ON or OFF (e.g. pushbuttons). In Arduino this is controlled and represented based on power output; 5V equates to a binary 1 (ON) while OV serves as 0 (OFF). This is subsequently represented in code as HIGH and LOW.

Analog signals are more refined, as their values can vary, expressing more than a discrete set of values. In the case of CSAW, potentiometers produce a voltage dependent on the amount of rotation (resistance) applied to the sensor. While analog pins on Arduino can also be used to function as GPIO/digital pins, their main purpose is to read analog sensors. On Arduino, pins that can read analog voltages are signified by the letter "A" – A0, A1, etc.

CSAW will mainly employ analog inputs through its 54 potentiometers, though the controller will also house 11 pushbuttons of digital input. Arduino Mega has upwards of 50 digital I/O pins but only 16 analog pins, requiring further analog input expansion through the use of multiplexers. These devices and their integration are discussed in further detail in Section 4.

Computers rely on digital representation of data to manipulate information, hence almost all microcontrollers have a built-in device in order to convert analog voltage on a pin into a digital number; an Analog-to-Digital Converter (ADC). CSAW's Arduino Mega has a converter allowing for 10-bit resolution to detect 2^{10} (1024) analog levels - i.e. it returns discrete values between 0 and 1023 (analogRead() reference, arduino.cc, 2019). This is based on a 5V supply voltage; the value range decreases using a smaller supply.

ADC resolution is generally quantified by the formula (wikipedia.org, 2019):

$$Q = E_{FSR} / 2^{M}$$

where Q = resolution in volts per step

E_{FSR} = the full-scale voltage range

^{M} = the ADC's resolution in bits

However, when dealing with converted digital values in Arduino (where values can be easily confirmed via the IDE Serial Monitor tool), the formula below is more appropriate for analog representation. This can be confirmed with a potentiometer, where turning the pot alters the amount of applied resistance and increases the voltage supplied, assuming a constant current (per Ohm's Law).

$$D = \frac{V_{adc} \times 2^M}{V_t}$$

where D = the converted digital value

V_{adc} = the voltage across the ADC

^{M} = the ADC's resolution in bits

V_t = the total voltage supplied (learningaboutelectronics.com, 2019)

e.g. A pot running off 5V turned approx. halfway (i.e. applying 2.5V across the ADC)

$$D = \frac{(2.5 \times 2^{10})}{5}$$

$$D = 512 \ (in\ Serial\ Monitor)$$

3. Development Process

Design for the CSAW controller follows a similar layout to commercial controllers and audio mixing consoles, containing a range of potentiometers and buttons to perform DAW tasks. The following section outlines the development and design processes for the project, in particular depicting the various stages of the control surface layout and consolidating a number of technical diagrams and blueprints which help summarize the project workflow, architecture and implementation requirements.

Figures accompany each subsection for easy reference, with full scale copies of all diagrams also available in Appendix C.

3.1 Project Planning

3.1.1 Gantt Charts

Work on the CSAW project took place from the start of June until mid-August 2019 over the course of 11 weeks. Tasks were divided into six separate stages as per the Gantt chart in Fig.10; a technical research and system design stage, three prototype stages establishing core functionality along with wireless capabilities and other project requirements, a system and user testing stage and finally, a stage of final checks and tests before the project submission.

Each of the stages depicted on the chart has their own respective Gantt chart detailing sub-tasks and associated dates. All charts were created using the Office Timeline Online application and are available in Appendix C.1 (officetimeline.com, 2019).

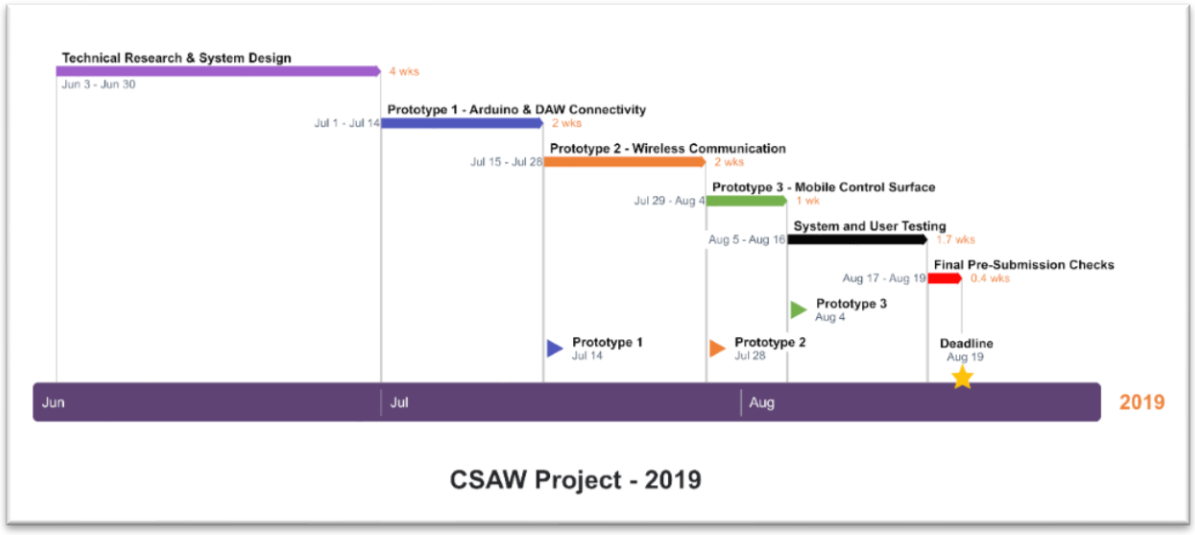


Figure 10. *CSAW Gantt chart*

A great deal of the project timeline was allocated to research and design. A substantial amount of time was dedicated to working with technologies and platforms where only a fundamental understanding was possessed so it was crucial to devote enough effort to accurately explore these areas in detail. This made the process of following the project's system development lifecycle (SDLC - Section 3.1.3) much easier as less time was spent re-reviewing requirements and design specifications after this initial timeframe. This timespan encapsulated everything from state-of-the-art review and technology research to requirement

specifications and design schematics and blueprints, on top of some fundamental Arduino work. CSAW's implementation took place over a month and a half of the timeline, with soft deadline prototypes every two weeks until the final deadline.

3.1.2 Functional and Non-functional Requirements

Functional requirements detail the key functions and behaviours of a system, while non-functional requirements outline general characteristics that can be used to judge the system (Functional and Non-Functional Requirements - Specifications and Types, 2019). While any project of this size could harbour a whole host of different requirements, the key requirements for the project can be quantified by Fig.11 below, consolidating both hardware and software requirements of the CSAW control surface. Each function/system behaviour is outlined as a functional requirement and then elaborated upon and detailed as a non-functional requirement.

More details on specifics of certain requirements (numerical ranges, controller operations etc.) are explored in later sections of this document.

Functional requirements	**Non-functional requirements**
The system must send the appropriate data when a condition is met - specific to sensor.	Discounting a minimal expected level of jitter for other pots, data should not be sent outside of the expected sensor/pot ranges.
The system must interact with the proposed application in a quick and timely manner.	Latency between control surface interactions and the actions in the DAW application must not exceed 10ms (noticeable delay).
The system must behave in an intuitive and user-friendly manner, based on other similar systems.	Operations (volume, EQ) should be labelled appropriately on the controller and perform exactly what is expected in the DAW.
The system must be able to transmit data over wired or wireless communication.	Two separate programs/patches are required for USB Serial and WiFi communication.

The system must not require a separate external power source to run at peak performance (but has the facility).	An integrated 12V battery pack is required in controller design for powering Arduino instead of USB.
The system must run as a self-contained unit as much as possible.	Outside of the necessary Arduino and Pd packages, no other libraries/applications should be required for controller operation.
The system should be easily adaptable and adjustable to expand functionality.	All Arduino and Pd code must be explicitly commented for easy expansion/adjustment.
The system must be easily deployable, not requiring components outside of the public domain.	Only open-source platforms and software must be used for implementation.
The system should be capable of replication based on the accompanying documentation.	All code/documentation must provide detailed info for replicating the CSAW project.

Figure 11. *CSAW Requirements*

3.1.3 Development Lifecycle

The project's development lifecycle follows Winston Royce's Waterfall model for engineering and software design (Royce, 1970). Principally for software-based systems, the Waterfall model was chosen for CSAW for a number of reasons – mainly due to the model's high-level design for using project deliverables, its suitability for short-period projects with a foreseeable deliverable date and the model's propriety to the project's clear, unambiguous requirements.

Separation of project activities were broken down into the model's key development phases:

- Requirements & Analysis: outlining CSAW's functional/non-functional requirements and analysing it alongside commercial control surfaces in the public domain.
- Design: designing the CSAW controller and architecture (hardware and software) based on the requirements and proposed technologies (Arduino, OSC etc.).
- Implementation: developing the control surface and integrating software using the available design blueprints to fulfil project requirements.
- Testing: debugging and testing the application.

- Maintenance & Operation: reviewing/evaluating the finished CSAW structure and establishing and documenting maintenance/installation/operation processes.

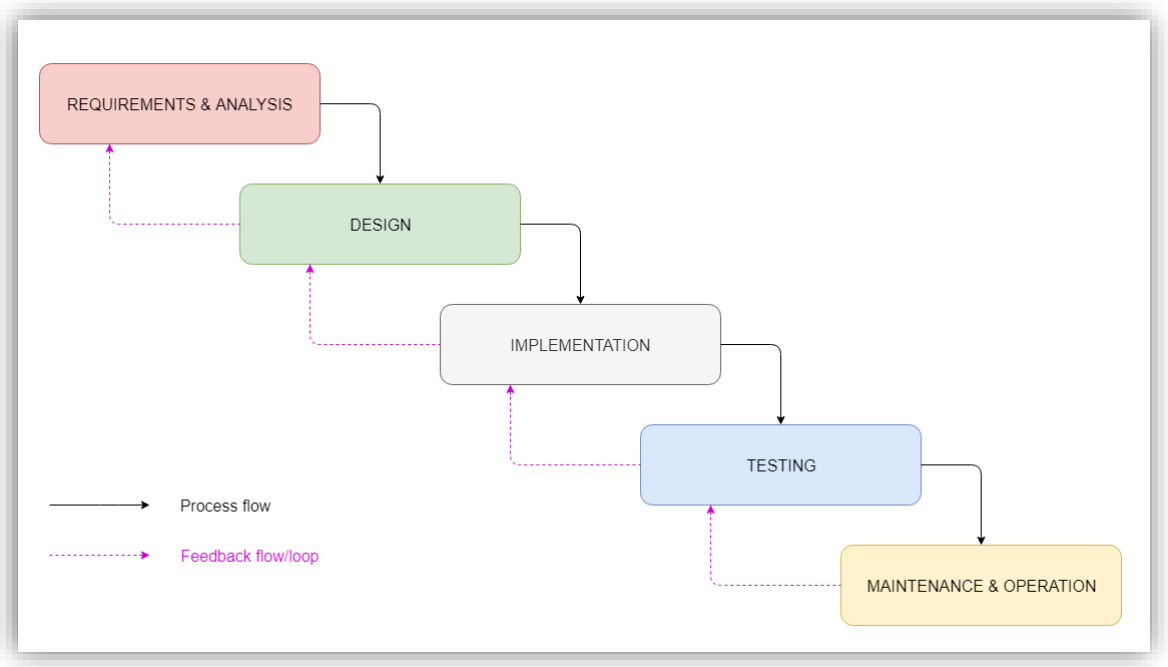


Figure 12. *CSAW Waterfall SDLC*

As Fig.12 suggests, CSAW's development process adhered to Royce's final Waterfall model with iterative feedback. Unlike the original model, which maintains progression from one phase to the next only when the preceding phase has been finalised, this model illustrates that feedback from phases can influence previous phases – for example, testing of code may bring implementation/design flaws to light. Since the project required prototypes at several stages, this was beneficial for reviewing and readjusting code, requirements and design specifications as the project progressed.

3.1.4 Required Materials

A list of the primary components and materials used in constructing the project can be found in Appendix C.3, along with approximate costs. Prices are correct as per 5 August 2019 and do not include tax or shipping charges. While certain materials listed are optional, and

quantities of others are more than what is required, everything listed in the materials table is recommended for recreating a similar control surface.

3.2 Blueprints & Technical Diagrams

3.2.1 System Interaction Diagrams

Architecture Diagram

Fig.13 displays the basic conceptual level components for the CSAW infrastructure. It gives a visual representation of the core functionality and components of the CSAW prototype and outlines the two methods for audio control in a DAW on a computer; transmitting the control signal via USB/wired connection (red arrow) and via the ESP8266/wireless connection (blue arrow) with the help of a regular WiFi router. A Pd patch running on the host computer relays controls from both methods to the DAW via Open Sound Control.

The diagram was created using the draw.io online diagram software, available at a larger resolution in Appendix C.4 (draw.io, 2019).

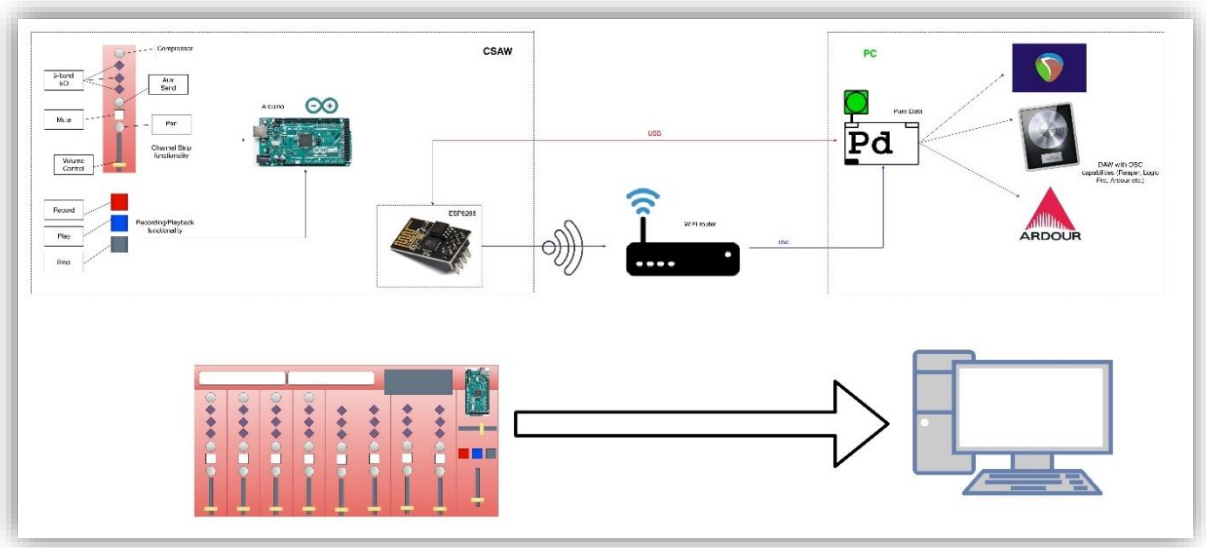


Figure 13. *CSAW Architecture*

Activity Diagram

The activity diagram below displays the general workflow of using the CSAW control surface, with support for choice between wired and wireless capabilities [Fig.14]. Both

methods follow the same general data flow as per the arrows and only disconnect at a few points in the diagram for function-specific actions. Ellipses represent the tasks or actions that can be performed while diamonds represent conditional statements or decisions (UML Activity Diagram - tutorialspoint.com, 2019).

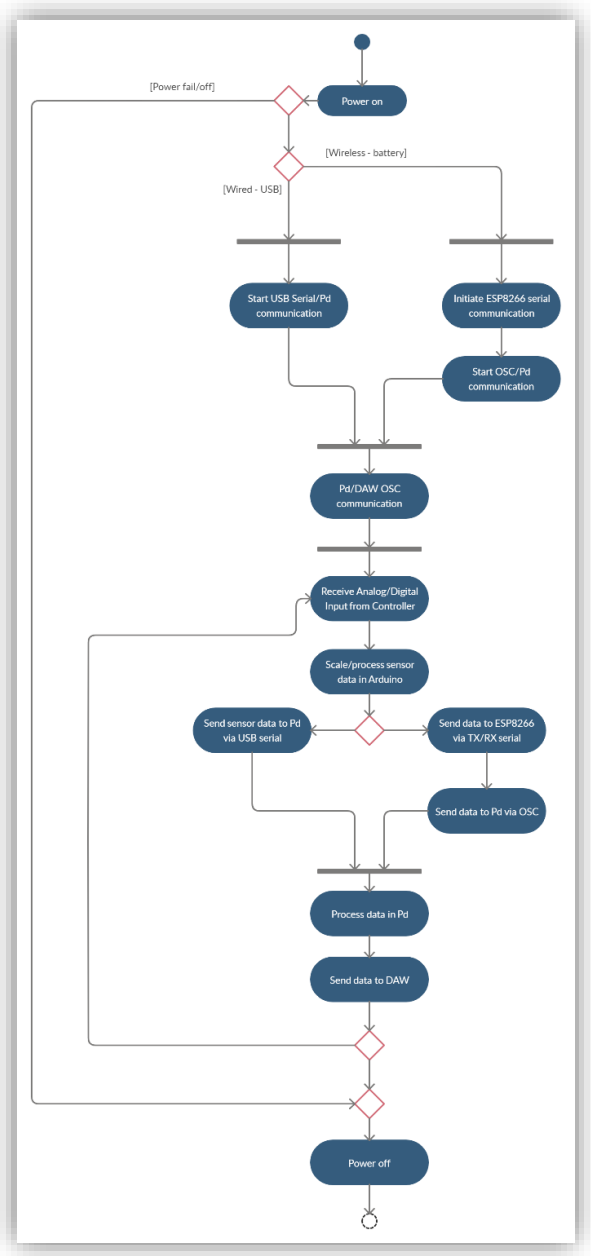


Figure 14. *CSAW Activity Diagram*

3.2.2 System Diagrams

Scale Diagram & Numbering Scheme

Several scale diagrams were hand drawn in the preliminary stage of the project, as outlines for prototypes. A list of proposed components was drawn up (some of which had already been acquired at this stage of the project) and size approximations for the desk were made based on measurements of these components. The scale drawing in Fig.15 below shows a mid-stage desk prototype (dimensions were later adjusted).

The diagram also outlines the numbering scheme for serial data sent from the Arduino Mega, per the red number next to each sensor. This signifies the starting point of the numerical range for each potentiometer (e.g. 3401 = 3401 to 3500), as well as the floats recognisable for pushbutton functions, to be understandable and processed appropriately by CSAW's Pd element. This remained unchanged from the final model.

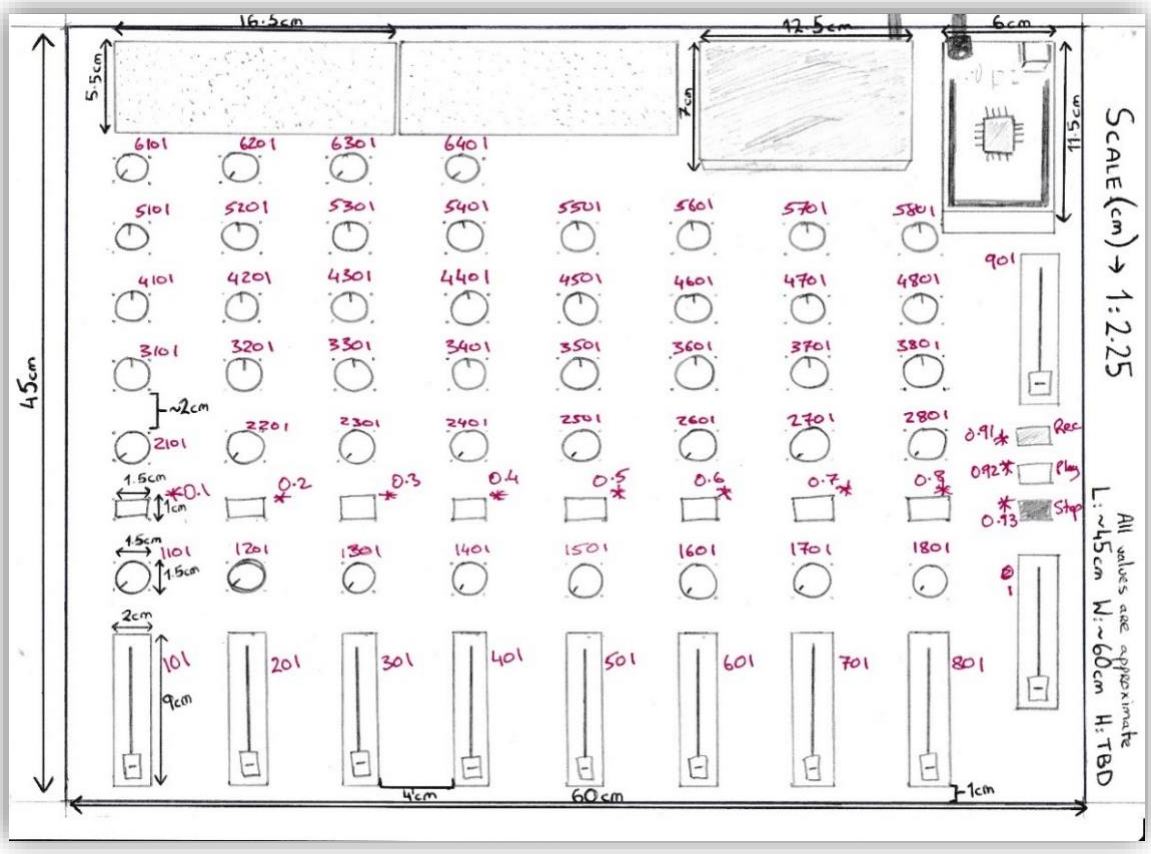


Figure 15. *CSAW Scale Diagram*

It was determined at an early stage to incorporate breadboards into the final prototype instead of soldering with printed circuit boards (PCBs) based on the recursive nature of the development and testing processes.

3D Model Representation

A 3D model was designed using the SketchUp 3D modelling design software, developed by Trimble Inc (sketchup.com, 2019). While this model was also altered over the course of the project, the core blueprint of the controller is the same. The layout of potentiometers and most other components remain unchanged.

The figures below display the model and the original SketchUp file can be found in the project repository and USB attached to this report.

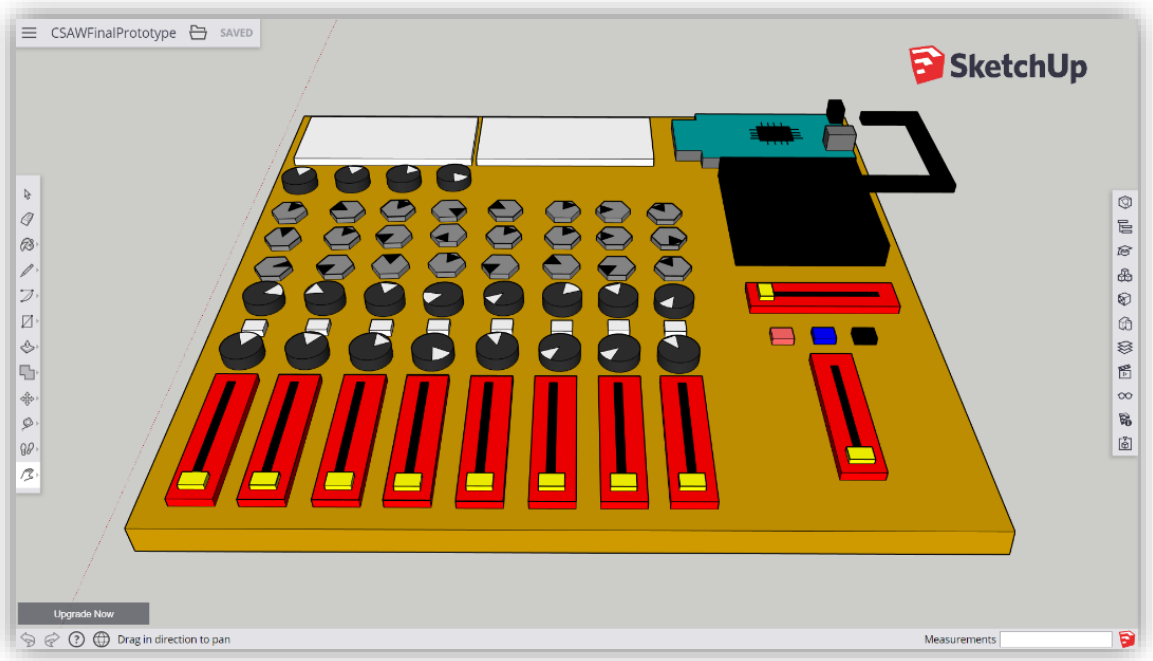


Figure 16. *CSAW 3D Render - Top view*

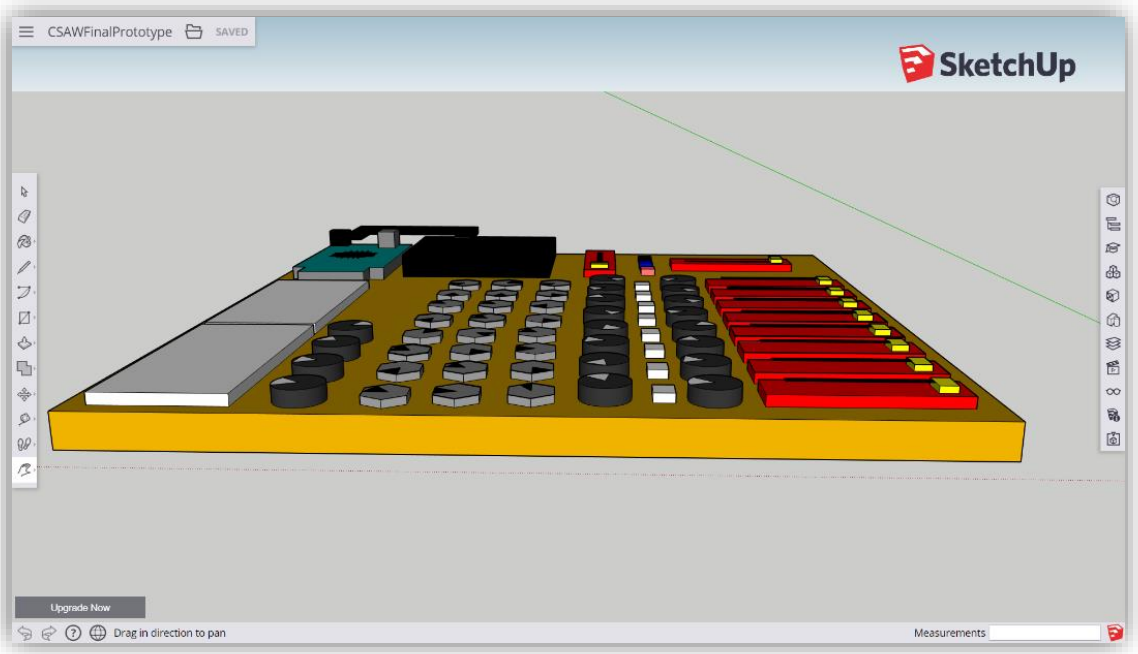


Figure 17. *CSAW 3D Render - Side view*

Final System/Measurements Diagram

The final system diagram was designed to aid construction of the physical desk. Fig.18 was the proposed final layout of the CSAW controller, however the positions of the Arduino and the battery pack were swapped during testing stages when issues were discovered receiving values between multiplexer select pins and Arduino's GPIO pins over large lengths of jumper-wire.

The accompanying schematic contains exact measurements of all key components on the board along with spaces for holes for jumper wires, padding, etc. Additional smaller breadboards keep pushbuttons and the ESP-01 separate.

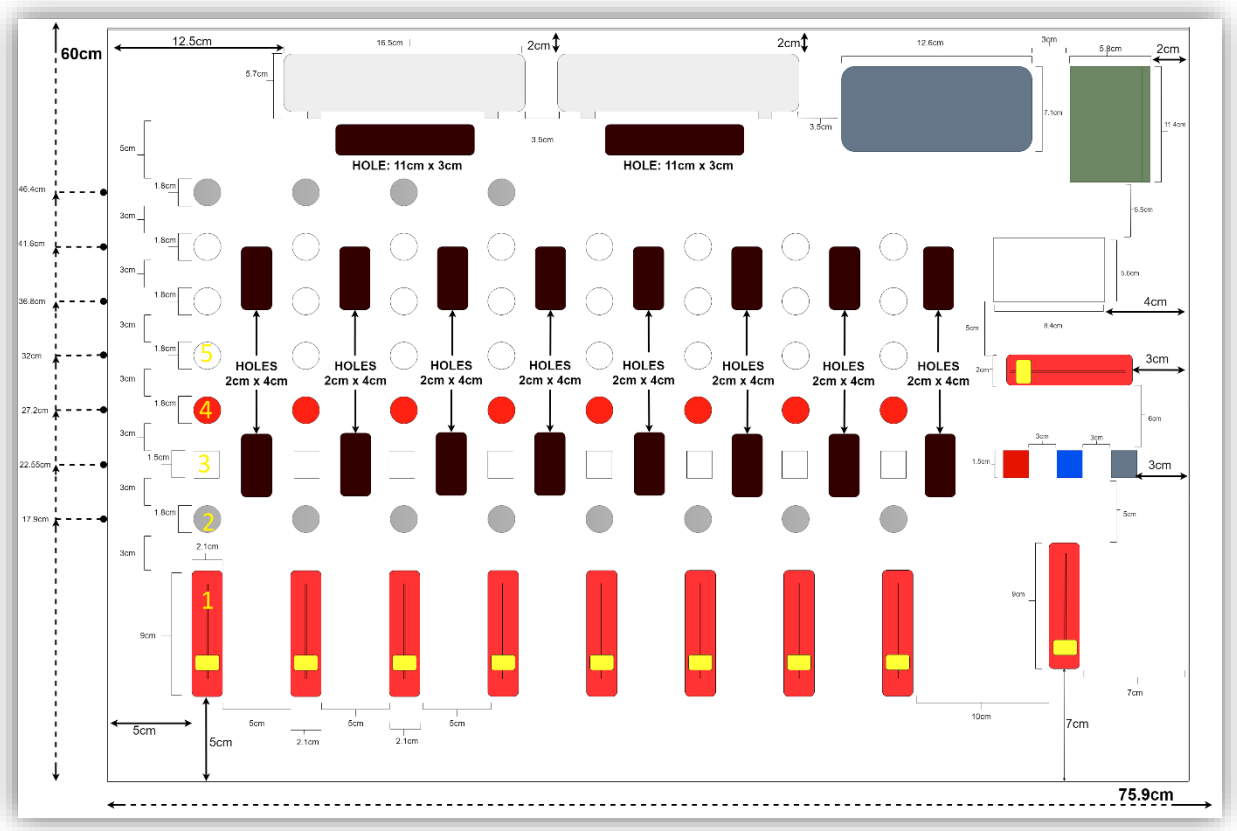


Figure 18. *CSAW Measurements Diagram*

3.2.3 Circuit Diagram & Schematic

Circuit Diagram

Circuit diagrams are graphical representations of an electrical circuit, outlining the primary physical components and their electrical connections using industry-standard symbols for universal understanding (University of Alicante - Circuit Diagram, 2019). Fig.19 uses CAD symbols – where wire contacts are established by circles/dots. The Arduino and ESP8266 components are depicted as integrated circuit boards with pin distinctions where appropriate.

Since circuit diagrams do not necessarily represent the physical arrangements of these components as part of the control surface, units are arranged in a human-readable manner, grouping resistors and muxes for channel strips as well as other functionalities (e.g. pushbuttons). All units share a common VCC and GND source from one of the several breadboards on the desk, originating from the Arduino Mega pins.

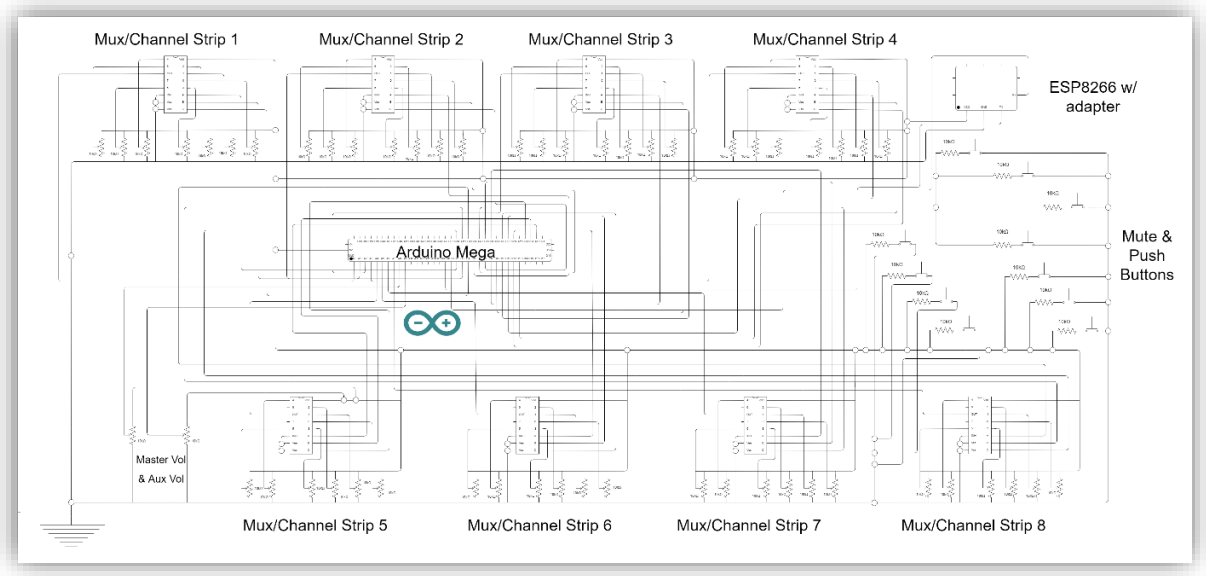


Figure 19. *CSAW Circuit Diagram*

Ohm's Law, one of the key governing laws of electricity, is integral to the understanding of all electronic circuits (wikipedia.org, 2019). Before building the above circuit, it was critical to establish the total resistance and required current draw for the CSAW project, given:

$$V = IR$$

All potentiometers and resistors used are of 10kΩ resistance. Generally speaking, pots between the range of 1kΩ and 50kΩ are considered more favourable as anything smaller takes too much current while anything larger takes longer to read or can be affected by values of other pots. Therefore, 10kΩ is the optimum value for resistors in a project of this size utilising many potentiometers, as many can be used before total current draw becomes an issue.

The maximum current draw for the ATmega2560 chip running on the Arduino Mega is 200mA across all pins, with a limit of 40mA per I/O pin. Assuming a 5V supply voltage (such as over USB), the maximum total current draw across all potentiometers/resistors in the CSAW circuit can be easily calculated:

Total capable resistance:

$$\frac{1}{R} = \frac{1}{10000} \times 65 = 0.0065$$

$$R = 153.85 = 154\Omega$$

$$V = IR$$

$$5V = I \times 154\Omega$$

$$I = 0.0325A = \underline{32.5mA}$$

Alternatively:

$$5V = I \times 10000\Omega$$

$$I = 0.0005A\ (0.5mA)\ [one\ pot]$$

$$0.5mA \times 65 = \underline{32.5mA}$$

This is less than the maximum limit across one I/O pin. Using a 12V power supply (either via DC power supply or controller battery pack), this increases to approx. 78mA, still less than half the maximum current draw of the Arduino across all pins. As resistors change value and the CD4051BE multiplexers act as switches for inputs, the exact current draw would also be much less than those forecasted here.

Schematic

In addition to the circuit diagram, it was essential to map out the physical arrangements of all analog sensors, Arduino & ESP, multiplexer and pushbuttons along with pin connections. This proved beneficial during the implementation stage of the project, where the diagram below was followed as a guide for wire connections between devices. The schematic was drawn up using the Fritzing open-source CAD software (fritzing.org, 2019).

A legend/key for pin connections can be found alongside a larger resolution copy of the diagram in Appendix C.10. All pin connections and components on the finished CSAW controller match Fig.20 below:

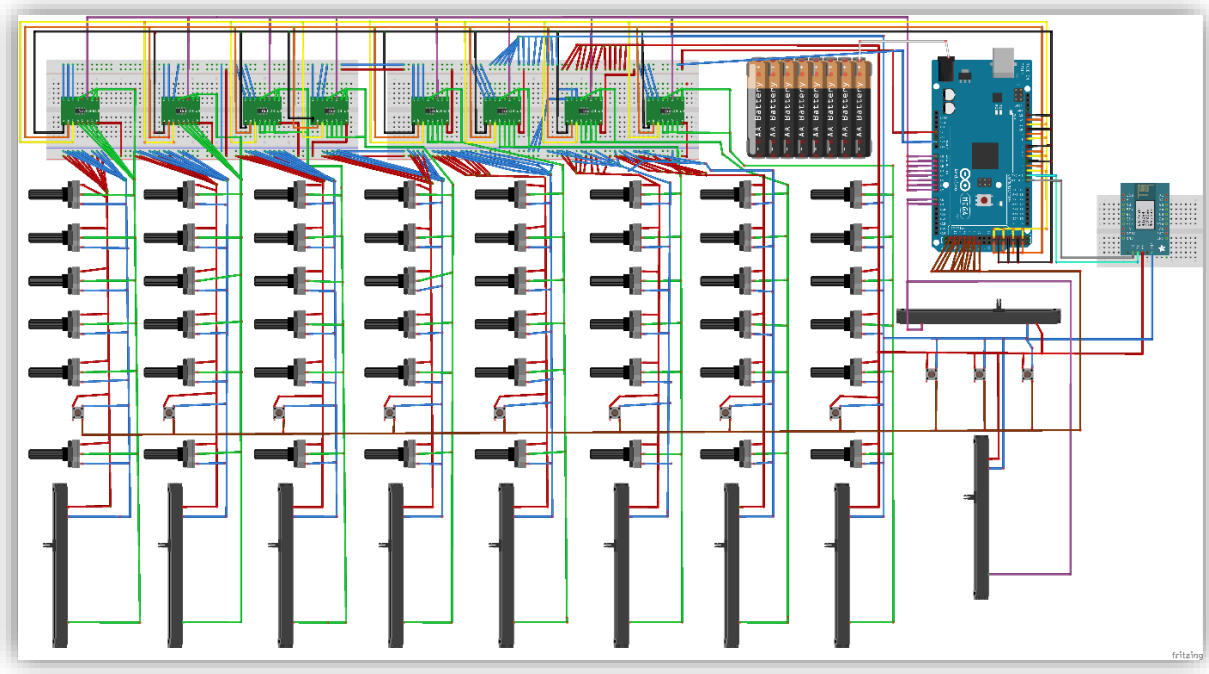


Figure 20. *CSAW Fritzing schematic*

A table specifying all pin connections is available in Appendix C.11. Certain Arduino pins were not used in case of serial comm or multiplexer issues allowing for easier replacement next to neighbouring pins. All ESP pin connections operate through an ESP-01 adapter module, acting as a voltage converter from the Arduino 5V power supply (see Section 4.2.1).

4) Analysis, Prototyping & Implementation

This section details the implementation process and prototypes of the CSAW control surface.

A note on Serial Data, libraries & Pure Data:

One of the goals of the project was to provide CSAW as an independent proof of concept, with minimum external libraries or firmware changes required for Arduino (refer to the project folder README file for library details). The intent was to transmit all data via serial communication where practicable. This involves additional scaling and processing before DAW mapping, but allows for a more adoptable solution, without installing and configuring surplus externals.

Additional configurations outside of WiFi/OSC libraries for Arduino are applied on the server side. Firmwares like Firmata prove useful for quick and easy analog/data detection from Arduino but ultimately require a lot of tinkering and expansion for multiplexer data and other project-specific functions (Firmata reference - arduino.cc, 2019). From experience, reverse-engineering a library's existing code is rarely efficient.

Arduino has many capabilities, including third-party OSC libraries, begging the question does Pure Data need to be utilised at all. While it may seem prudent to cut out Pd and simply process and send OSC messages directly to a DAW from Arduino/ESP, this introduces a greater margin of error and is tougher to debug, giving no indication that the application is working effectively.

For CSAW demonstration, Pd aids data processing, gives a visual representation of data transmission and provides uncomplicated troubleshooting. It also allows for easier OSC message reconfiguration and mapping should other DAW parameters be introduced, instead of constantly re-uploading Arduino code. Any potential latency between Pd and a DAW is minimal or unrecognisable, due to running on the same host.

4.1 Wired Prototype – Prototype A

Prototype A consisted of a control surface whereby Arduino was powered and communicated with PC software over the USB port. No wireless capabilities were introduced at this point. Circuitry and software for each functionality of the desk was developed independent of one another and consolidated at a later date to make a working channel strip.

All Arduino & ESP code, Pd patches, examples and more can be found on the accompanying USB and at the CSAW project's GitHub repository. Code/patch excerpts can be found at a higher/larger quality in Appendix D.

4.1.1 Functionality

Initial Functions/Volume Control

The first function was to implement volume control for CSAW. This involved mapping and scaling a potentiometer's values to a volume fader running in a DAW (REAPER for initial tests). A simple circuit was established using Arduino and a 10KΩ linear slide potentiometer as per the images below:

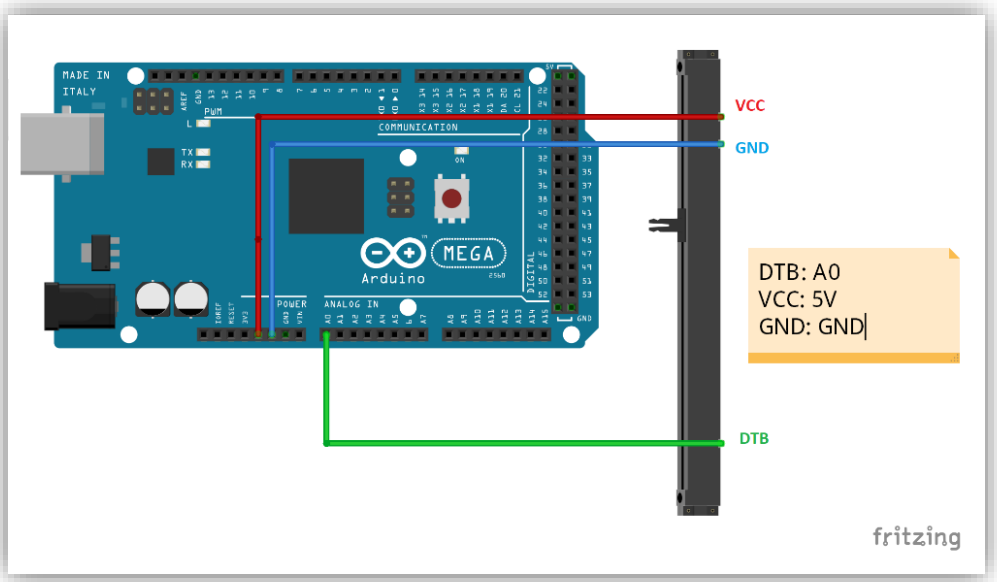


Figure 21. *Slide Potentiometer example (fritzing)*

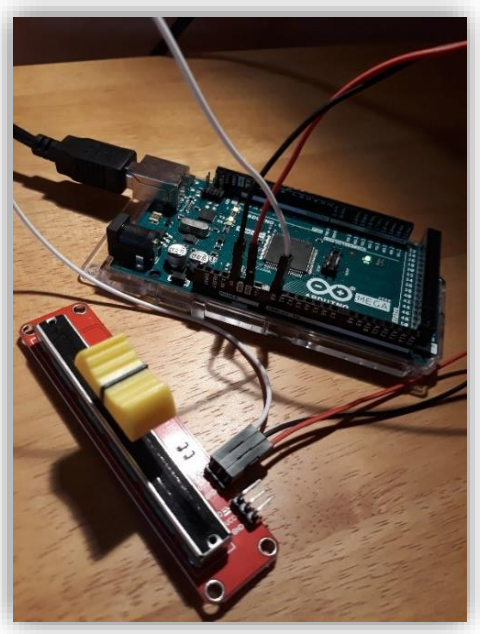


Figure 22. *Slide potentiometer example (photo)*

In the Arduino sketch, values were scaled between 0 and 100 for easier readability before being written to Arduino's serial communication bus (`Serial.write()` - arduino.cc, 2019). REAPER occasionally crashes with a flood of incoming OSC messages, so a delay was introduced (further rate handling is discussed later in this report). Data from the bus was received in Pd over serial (USB), where it was further scaled as a float between 0 and 1 so DAW volume control actions could understand.

This value was then sent over the (local) network from Pd to REAPER as an OSC message, achieved via an OSC software control surface in REAPER settings. The DAW is configured to listen for OSC messages on a specific IP and network port (the same local computer IP and port specified in the Pd patch). Most importantly, these messages were configured to be bound with REAPER's built-in actions list and to be used for plugin/FX learning. Finally, an action was created in REAPER's action list menu for the "Track: Set volume for track 01 (MIDI CC/OSC only)" action. When selecting to add a shortcut for the action the "/test" OSC message was detected.

The Arduino sketch and Pd patch (with additional comments) as well as a screenshot of DAW OSC settings for a single volume fader function can be seen in the figures below:

Vol_Control

```
/***Vol_Control***/
// A simple volume control sketch - scales and writes pot data to Serial bus

// Specify analog input pin and initialise its value
int sensorPin = A0;
int sensorValue = 0;

void setup() {
  Serial.begin(115200); // Baudrate determines data rate in bits per second for serial transmission
}

void loop() {
  sensorValue = analogRead(sensorPin); // read pin value
  sensorValue = map(sensorValue, 0, 1023, 0, 100); // scale it between 0 and 100
  Serial.write(sensorValue); // write the value to serial
}
```

Figure 23. *"Vol_Control" Arduino sketch*

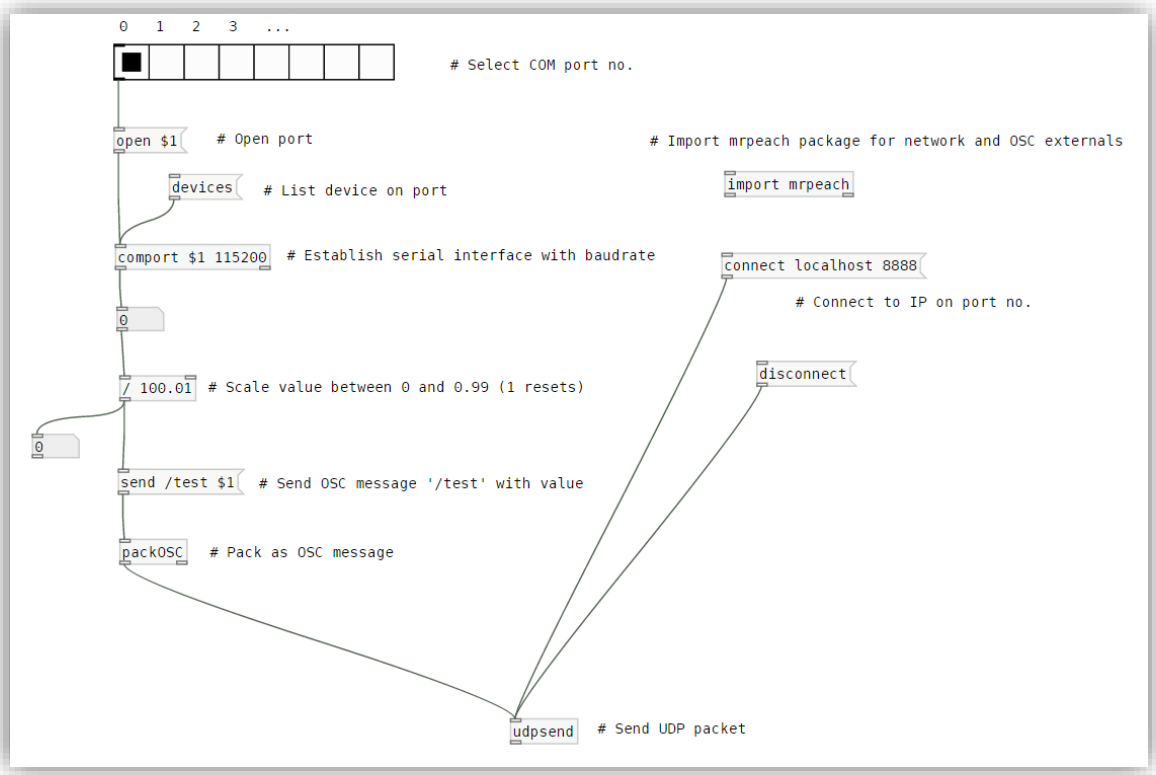


Figure 24. *"Vol_Control" Pd patch*

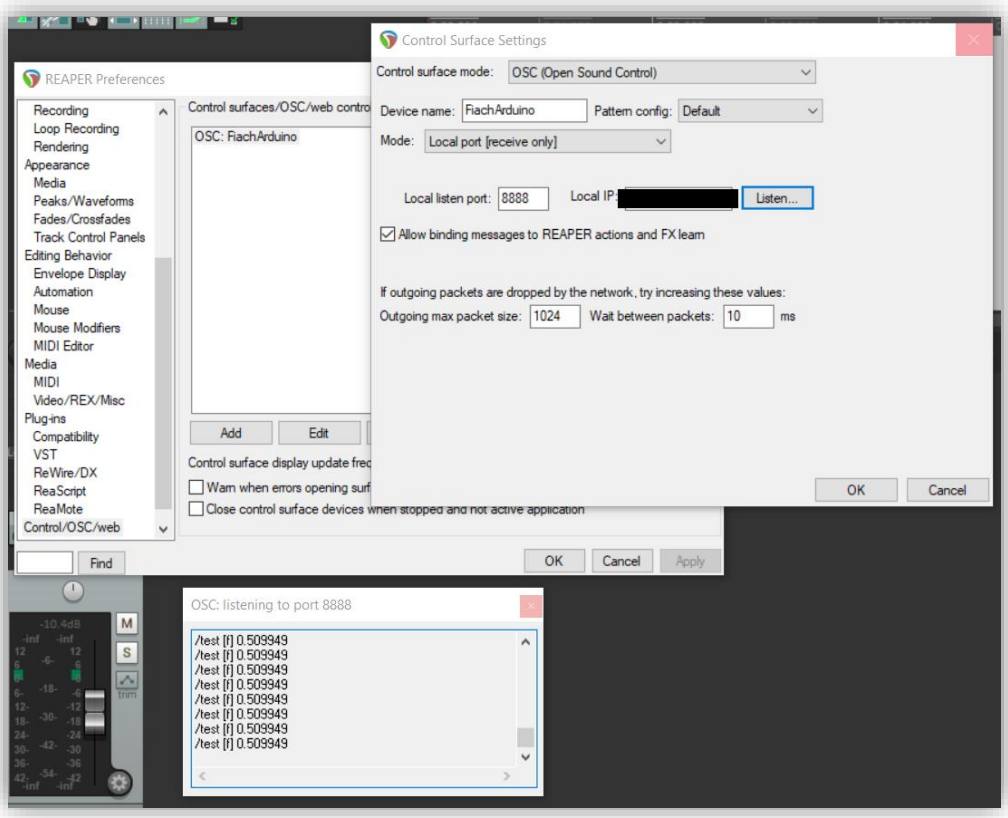


Figure 25. *REAPER OSC software control surface settings*

It was soon observed when testing with multiple potentiometers that transmitting pot data in this manner would not prove feasible. Due to the nature of Arduino's `Serial.write()` function, data is sent to the serial port as bytes – meaning a maximum limit of 255 (the largest potential decimal value of a byte). The `Serial.write()` function was used for data recognition in Pd, as `Serial.print()` and `Serial.println()` are not recognised properly (Serial reference - arduino.cc, 2019). This would only allow for two potentiometers ranging 0-99 and 100-199, unless ranges were scaled down - greatly reducing the granularity of each pot. This was unacceptable and an alternative solution was needed.

It was considered resolving this through float conversion functions or passing through buffers and then deciphering them within Pd. Fortunately, there was a superior fix through the use of an existing external repository on GitHub of Pd abstractions - Arduino_Pd (alexdrymonitis/Arduino_Pd, 2019). The `print_serial` patch allowed for data passed through the `Serial.print()` or `Serial.println()` functions to be recognized by Pd, permitting an infinite scope of number ranges and the application of number ranges for

parameters. Each range now acts as an identifier for data per the numbering scheme in Appendix C.6.

N.B. The `print_serial` patch from the Arduino_Pd repository is required to run the main CSAW patches and most examples.

This process was duplicated for multiple faders, routing VCC and GND for each pot to Arduino via a breadboard and connecting pots to separate analog pins. Each pot was connected individually at first and the resulting message mapped to a volume action in REAPER. An example mapping 8 volume faders to tracks can be found in the project repository. The final prototype patch includes the master and aux-send volume controls (each with their own ranges) and altered the setup slightly through grouping by channel strip instead of functionality.

In general, this same logic of receiving data from the Arduino serial bus, processing and scaling it appropriately in Pd and then forwarding it to the DAW was applied to other operations – with independent processing based on functionality.

Panning / Aux-Sends

A very similar process was applied to panning functions for each channel strip – reading the value from a potentiometer (this time a rotary pot). The only differences involved sending different OSC messages and mapping these to different actions in REAPER. An example for this can be found in the repository/on the USB.

Auxiliary sends, however, required some additional work. Using REAPER as the test DAW, it was discovered that there are no built-in actions to control the send value/level from a track to an auxiliary track via OSC/MIDI. Therefore, a workaround was established.

Separate tracks were created specifically as sends to the main aux-send. The “master send” checkbox was unchecked so as not to be output to the master track. Therefore, CSAW’s send potentiometers could then be mapped to the volume actions for each of these new tracks. For example, the send pot for channel strip 1 maps to the “Track: Set volume for track 11 (MIDI CC/OSC only)” action (a blank track was introduced for track 10 for easier alignment).

Figure 26. *Aux-Send & Volume tracks in REAPER*

The issue with this approach is that send tracks must remain fixed after the main channel tracks, as moving them around within the session cause OSC messages to interact with the wrong tracks in the DAW. For example, moving aux-send track 3 from its position of track 13 to the right of volume track 3 renames it as track 4. As a result, all other tracks are renamed and the OSC message originally configured to control track 13 volume now adjusts the volume for the preceding track and so on.

Equalization

A 3-band equalizer (EQ) on each channel of the CSAW controller was constructed, by way of a default DAW plugin. In REAPER's case, this was the ReaEQ plugin (ReaPlugs VST FX Suite, 2019). This was achieved using plugin FX learn capabilities, allowing for easy mapping of controls to plugin parameters.

The EQ is based on the mono input channels for the Behringer Xenyx Series – LOW, MID and HIGH. Turning potentiometers clockwise/counter-clockwise raises/lowers the gain for the corresponding band, where central positions for each pot equate to flat frequency responses for the respective ranges (Välimäki & Reiss, 2016).

In keeping with the reference EQ, these bands correspond to:

- LOW: a low shelving filter with a fixed cut-off frequency of 80Hz.
- MID: a band pass filter with a fixed cut-off frequency of 2500Hz.
- HIGH: a high shelving filter with a fixed cut-off frequency of 12000Hz.

Each channel also has a fixed fourth band of a low-cut filter at 75Hz. Normally associated with gain for analog desks, this is enabled in CSAW by default. Boosts and cuts are regulated to +/- 12dB as opposed to Xenyx's 15dB settings due to ReaEQ plugin parameters. Scaling alterations were made in Pd to prevent the plugin from exiting these ranges.

All bandwidths are fixed to default 2 octaves per the plugin. Excerpts of how this 3-band EQ was integrated can be found below with full code available in the project repository.

ThreeBand_EQ

```
/***ThreeBand_EQ***/
// Three-band EQ control using 3 potentiometers : 3 analog pins

// Define & initialise analog pins
int pot1 = A0, pot2 = A1, pot3 = A3;
int pot1val = 0, pot2val = 0, pot3val = 0;

void setup() {
  Serial.begin(115200); // baud rate
}

void loop() {
  // Read values for each pot and scale between unique number range
  pot1val = analogRead(pot1);
  pot1val = map(pot1val,0,1023,1,100);

  pot2val = analogRead(pot2);
  pot2val = map(pot2val,0,1023,101,200);

  pot3val = analogRead(pot3);
  pot3val = map(pot3val,0,1023,201,300);

  // Print all values to serial where Pd can understand via 'serial_print'
  Serial.println(pot1val);
  Serial.println(pot2val);
  Serial.println(pot3val);
}
```

Figure 27. *"ThreeBand_EQ" Arduino sketch*

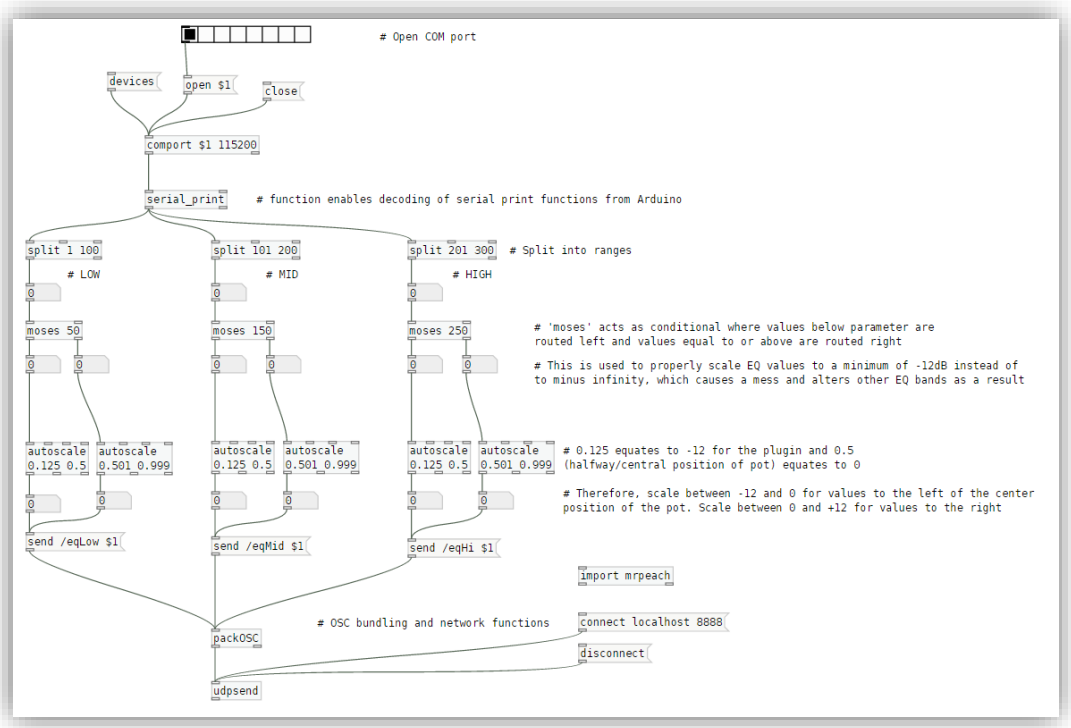


Figure 28. *"ThreeBand_EQ" Pd patch*

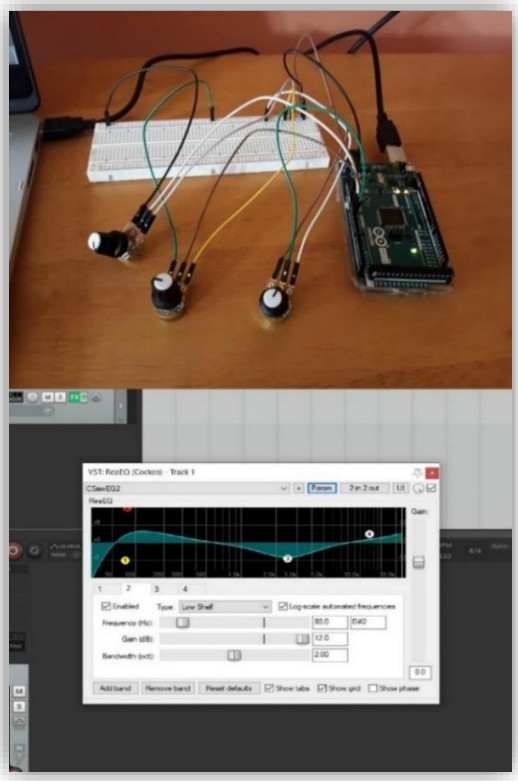


Figure 29. *Three Band EQ example with corresponding ReaEQ parameters*

Compression

Compression parameters for CSAW are based on the renowned one-knob compressor for the Yamaha MG Series audio mixers (Yamaha, 2014). Compressors typically have a number of different parameters, so it was important to implement a practical solution that carefully altered these parameters with respect to one another using a single potentiometer. Use of a one-knob compressor is ideal for this type of project, giving a more compact, centralized compression control.

Rotating the corresponding pot clockwise increases overall digital compression on the track through adjusting several dynamic processing parameters (iZotope & McLaughlin, 2014). This commits changes to the DAW's default compression plugin (ReaComp for REAPER) as close as possible to the MG compressor:

- Threshold: the level at which compression begins, i.e. where signals above this threshold are affected. The MG compressor reduces from +22dBu to -8dBu but since this is barely noticeable when translated to a DAW plugin (0.0dB is the default "uncompressed" threshold in ReaComp) this was scaled down. As a result, threshold reduces from 0.0dB to -30dB as the pot turns.
- Ratio: the amount of dynamic processing applied to a signal once it goes over the threshold. This increases from 1:1 to 4:1 as per the MG series user manual.
- Output Gain: the overall output volume after compression is applied. This increases from 0dB to +7dB per MG documentation.
- Auto Make-up Gain: automatically attempts to normalize the compression output gain. This option is enabled.
- Attack: how quickly compression is applied once the signal passes the threshold. This is fixed at 25ms per the MG compressor settings.
- Release: how quickly compression stops/drops once the signal passes the threshold. This is also fixed - at 300ms.

Similar to the Xenyx model that its layout is based on, CSAW has compressors on its first 4 channels only. For a control surface controlling 8 mono channels however, this was a creative choice based on limited potentiometers and proof of functionality for the project.

CSAW's compression process was carried out per the code/screenshots below.

OnePot_Compressor

```
/***OnePot_Compressor***/
// One knob compressor based on Yamaha MG series
// One potentiometer controls compressor ratio, threshold and output gain

// Define and initialise pin and value
int sensorPin = A0;
int sensorValue = 0;

void setup() {
  Serial.begin(115200); // baud rate
}

void loop() {
  // Read value, scale between 0 and 100 and send to serial
  sensorValue = analogRead(sensorPin);
  sensorValue = map(sensorValue, 0, 1023, 0, 100);
  Serial.println(sensorValue);
}
```

Figure 30. *"OnePot_Compressor" Arduino sketch*

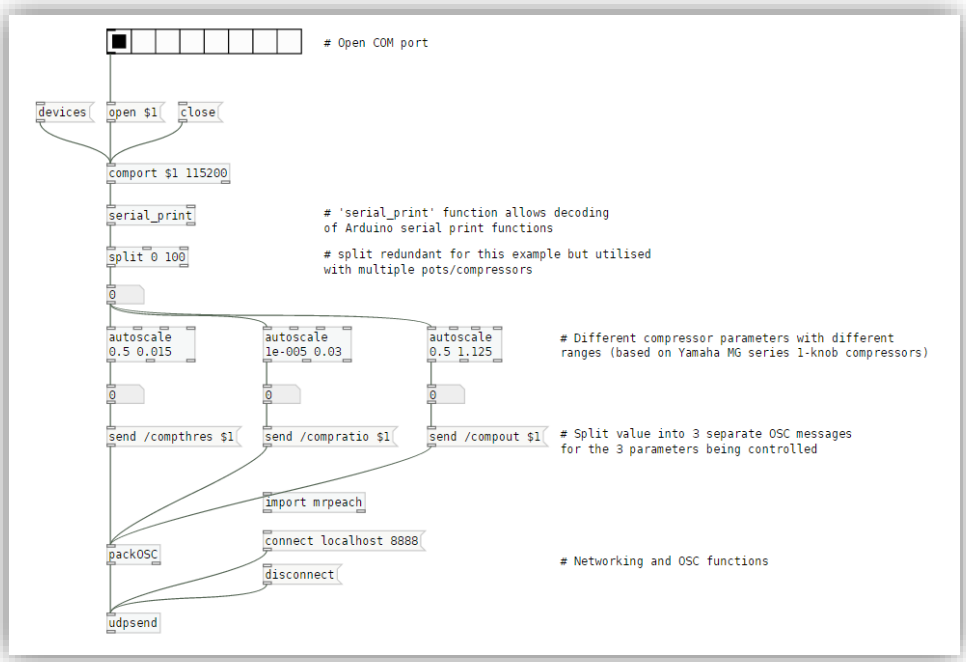


Figure 31. *"OnePot_Compressor" Pd patch*

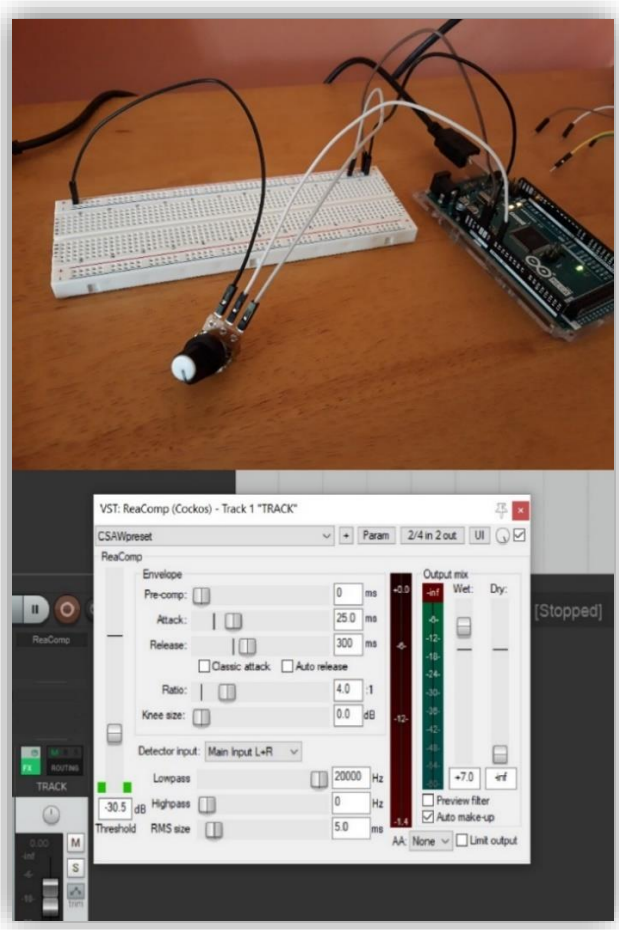


Figure 32. *Full Compression example with ReaComp plugin*

Transport & Solo buttons

The only functionality that uses digital signals, solo and transport processes were carried out in the same manner as basic pushbutton examples. The code is adapted from Arduino's Switch digital I/O example, expanded for multiple digital inputs and available from the IDE examples tab (Switch - arduino.cc, 2019). It was important to commit changes only once, with no room for noise/error so switch debouncing was included over a timeframe of 200ms with each toggle. Button states were only printed to serial on toggle, cutting down on extraneous messages being sent to Pd and REAPER.

Unlike the potentiometers seen so far, which acted as variable resistors, pushbutton functionality required separate resistors to maintain consistent readings. For this project, resistors were placed connecting pushbuttons to the ground connection, acting as "pull-down" resistors (electronics-tutorials.ws, 2019). This establishes a known state for a signal and

prevents the circuit from “floating”, i.e. having an indeterminate voltage. 10kΩ resistors were used, in keeping with the other potentiometers in the project.

An example Arduino sketch and Pd patch can be found below, showing the process for sending record, play and stop OSC messages from 3 separate buttons.

Transport_switches

```
/***Transport_switches***/
// 3 button input switches that print specific numbers to serial on press for transport functionality

// Initialise pins used, state variable, previous reading variable and current reading
int chan1PIN = 2, state1 = LOW, previous1 = HIGH, reading1;
int chan2PIN = 3, state2 = LOW, previous2 = HIGH, reading2;
int chan3PIN = 4, state3 = LOW, previous3 = HIGH, reading3;

// Time variables for debouncing (ensures only a single signal is sent)
long time = 0, debounce = 200;

void setup()
{
  Serial.begin(115200);
  pinMode(chan1PIN, INPUT);
  pinMode(chan2PIN, INPUT);
  pinMode(chan3PIN, INPUT);
}

void loop()
{
  // read pin, toggle its state and print number to serial line
  // set time to time Arduino is active and change previous reading variable to current state
  reading1 = digitalRead(chan1PIN);
  if (reading1 == LOW && previous1 == HIGH && millis() - time > debounce) { Serial.println(0.91); time = millis(); }
  previous1 = reading1;

  reading2 = digitalRead(chan2PIN);
  if (reading2 == LOW && previous2 == HIGH && millis() - time > debounce) { Serial.println(0.92); time = millis(); }
  previous2 = reading2;

  reading3 = digitalRead(chan3PIN);
  if (reading3 == LOW && previous3 == HIGH && millis() - time > debounce) { Serial.println(0.93); time = millis(); }
  previous3 = reading3;
}
```

Figure 33. *"Transport_switches" Arduino sketch*

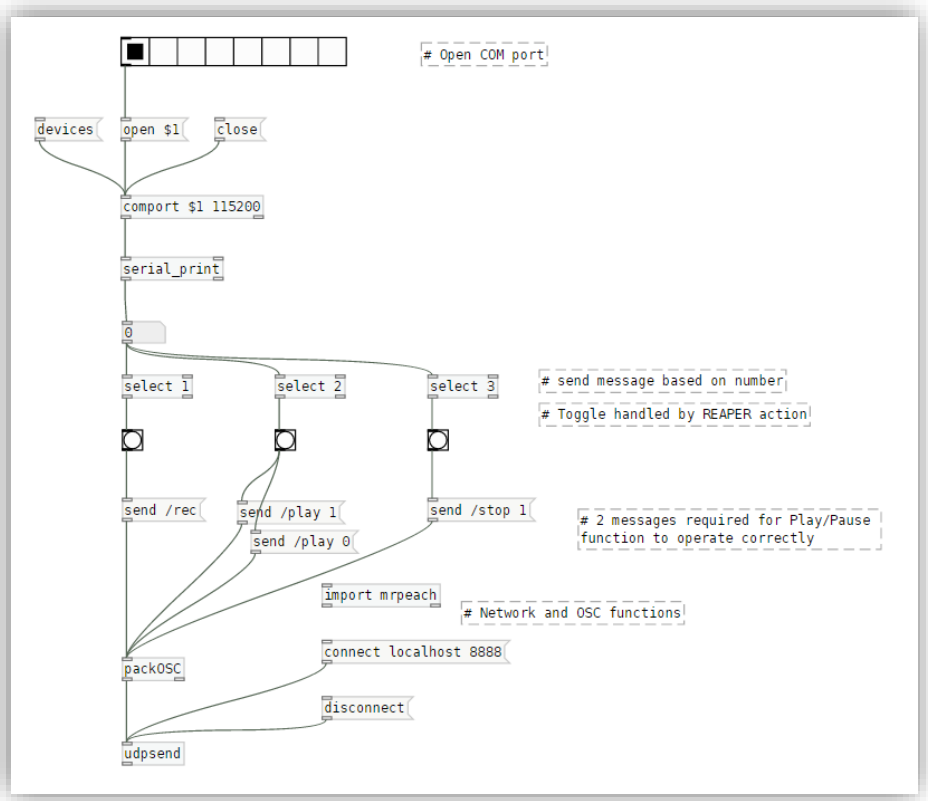


Figure 34. *"Transport_switches" Pd patch*

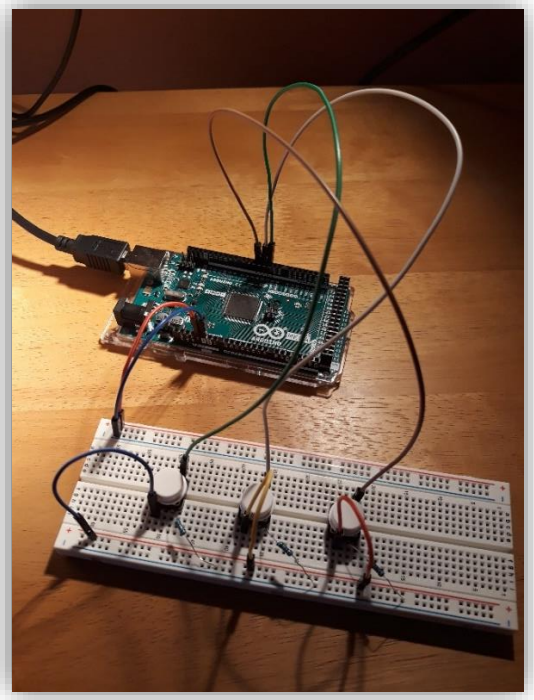


Figure 35. *Playback switch buttons*

This process was replicated for each solo button, again sending independent values from Arduino and separate OSC messages. Unlike the potentiometer ranges, float variables were used instead of integers so as not to clash with the predefined numbering scheme.

4.1.2 Multiplexing

As the number of analog inputs was much larger than the number of pins available on the Arduino board, multiplexing techniques were required. Multiplexing (or "muxing") is a technique used to combine multiple analog or digital signals into a single signal transmitted over a shared channel. This is achieved by way of a multiplexer ("mux"), a device that selects between these inputs and forwards it to the shared output line (Multiplexer Breakout Hookup Guide, 2019). Multiplexers come in several classifications, typically falling into the range of having 2^n input pins.

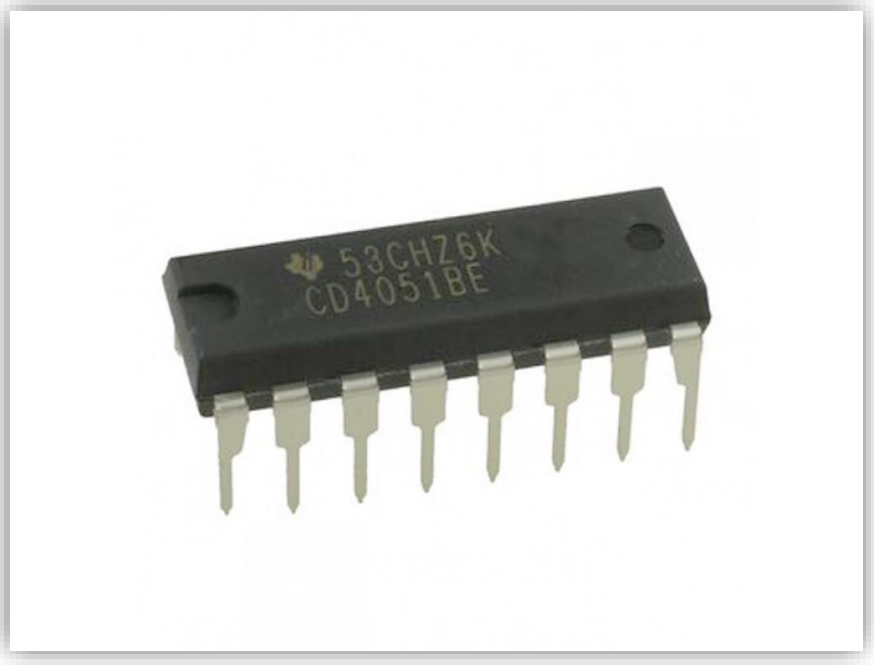


Figure 36. *Texas Instruments' CD4051BE 8-channel multiplexer (elcoteam.com)*

A number of multiplexers were tested and integrated into the CSAW project for this reason. The Arduino Mega has 16 fixed analog inputs, not nearly enough for the 54 pots that were

envisioned for the project. CSAW makes use of several CD4051BE 8-channel analog muxes, developed by Texas Instruments, allowing for 8 individual analog inputs per analog pins on the board (Texas Instruments, 1998). Originally, 74HC4067 16-channel muxes were considered to cut down further on analog inputs, however these were dismissed for the 8-channel muxes for easier management on a channel to mux basis (nexperia, 2015). Key multiplexer features can be found in Appendix B.3.

The CD4051BE has the following pins (may be labelled differently based on manufacturer):

- VDD: positive supply voltage
- VSS: ground supply voltage
- COM OUT/IN: common output pin (input if used as demultiplexer)
- CH0 - CH7: input pins to be routed to Z output
- A, B, C: select input pins used to choose the input to be routed
- INH: enable pin to make the mux operate
- VEE: negative power input

The process of deciding which input was routed to the output on the mux was based on the device's select pins. These are connected to digital GPIO pins on the Arduino. In the same vein as a binary truth table, a logic value of L or H is applied to each selector, where L signifies a "low" voltage (between 0 and 2V assuming the mux is powered at 5V) and H means a "high" voltage (3-5V). This is easily decided in code using Arduino's `digitalWrite()` function, to write HIGH or LOW to each pin (digitalWrite - arduino.cc, 2019). The combinatorial output of these selectors specifies which input is routed to the single Z output.

Outputs for the 8-to-1 multiplexer used for channel routing in CSAW are chosen based on 3 select pin values, identified by the table below.

C	B	A	OUT/Channel
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

Figure 37. *8-channel multiplexer truth table*

The excerpt below outlines Arduino code for utilising a multiplexer for channel inputs. The full example can be found on the USB/project repo.

```
void loop()
{
  // Loop through first seven of the eight pins.
  for (byte pin=0; pin<=6; pin++)
  {
    selectMuxPin(pin); // Select one at a time
    int inputValue = analogRead(A0); // and read Z
    inputValue = map(inputValue, 0, 1023, start, start+99);
    Serial.println(inputValue);
    start = start+1000;
  }
  Serial.println();
  delay(100);
  start = 101;
}

// The selectMuxPin function sets the S0, S1, and S2 pins
// accordingly, given a pin from 0-7.
void selectMuxPin(byte pin)
{
  for (int i=0; i<3; i++)
  {
    if (pin & (1<<i))
      digitalWrite(selectPins[i], HIGH);
    else
      digitalWrite(selectPins[i], LOW);
  }
}
```

Figure 38. *Excerpt from "Channel_Strip" Arduino sketch*

One channel strip for the control surface was then routed through a multiplexer as follows:

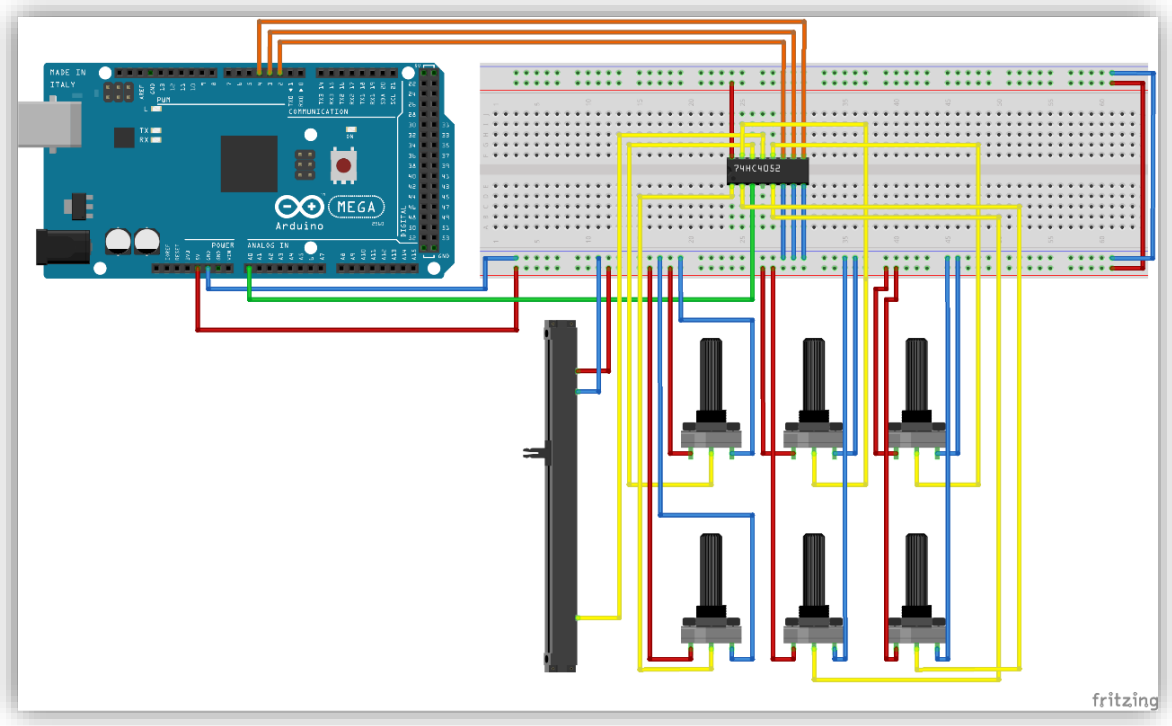


Figure 39. *Routing 7 analog inputs using CD4051BE multiplexer*

The output pin of the mux was connected to A0 on the Arduino, thereby extending the amount of analog inputs from 1 to 8 (or 7 in this case since 1 input was not being used). Since muxes act in a state of either analog or digital but never a combination of both, the solo & playback/recording pushbutton digital inputs are not included in any multiplex circuitry (though could potentially use their own independent mux if required).

This process was applied for all remaining analog inputs; routing channel controls to other multiplexers at the same ratio of 1 channel strip:1 mux. Appendix C.10 contains a full schematic of multiplexer and pin connections.

4.1.3 Final A Prototype

All functionality was consolidated with multiplexer capabilities into a single Arduino sketch and accompanying Pd patch. Screenshots of the files are available in Appendix D.6.

The final code is available in the project GitHub repository and on the USB accompanying this document under the “Prototype A” directory.

4.2 Wireless Prototype – Prototype B

Prototype B adapted its predecessor by incorporating data transmission over WiFi. Instead of communicating data directly to Pd via USB serial, CSAW makes use of the ESP-01 to relay OSC messages to Pd (Espressif Systems, 2018). Power for Arduino was now provided by a 12V DC supply.

4.2.1 ESP8266 Configuration

Before use, it is suggested to flash the newest firmware to the ESP chip. This requires some additional components – specifically a USB to TTL Serial converter adapter for coding the chip. It is also worth noting that the USB TTL converter and ESP8266 must share a common GND for uploading code as some guides do not specify this.

The adapter is inconsistent for providing enough power to run ESP, therefore a 3.3V power source for the module is recommended. Breadboard power supply adapters, like the MB102, are useful for configurations, converting higher voltage supplies to power the chip (Opencircuit, n.d.). If necessary, the power supply for powering CSAW's Arduino component could instead be plugged into the MB102 – splitting power to 5V for Arduino Vin and 3.3V (ESP) on two separate sides of a breadboard. Instead, an ESP-01 adapter module was used for the finished circuit - converting the 5V power provided to other components into 3.3V.

A full list of materials for the project can be found in Appendix C.3. While not necessary, it is recommended to acquire a breadboard adapter for ESP as well as multiples of certain components; in particular the MB102 adapters which tend to have erratic pins, requiring undue manipulation to provide power after extended use.

Whether flashing firmware or uploading code to ESP over the course of the project, the same pin connections were followed:

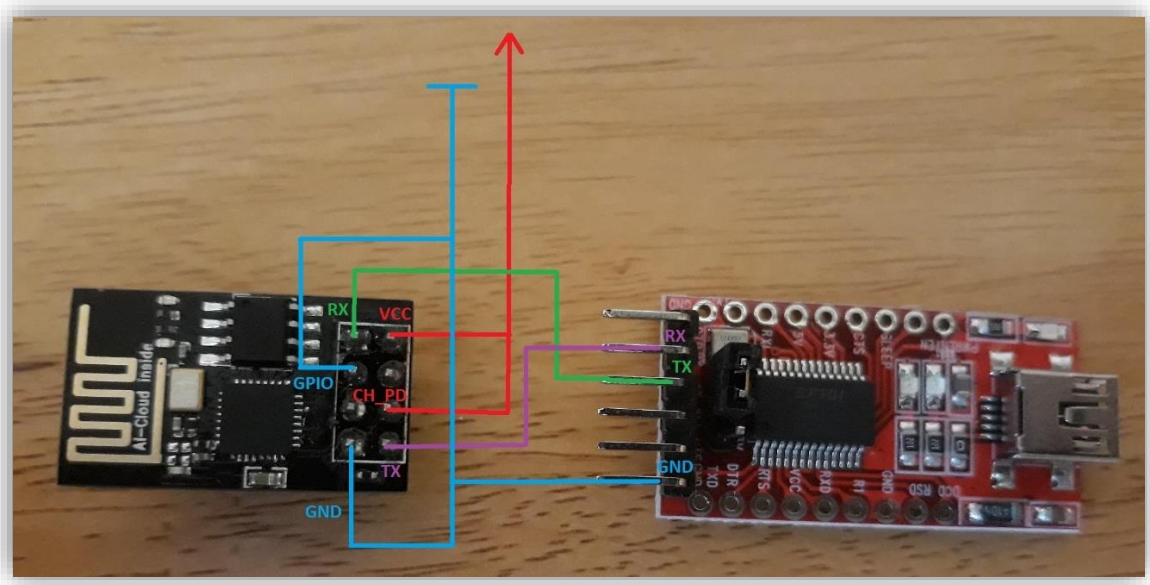
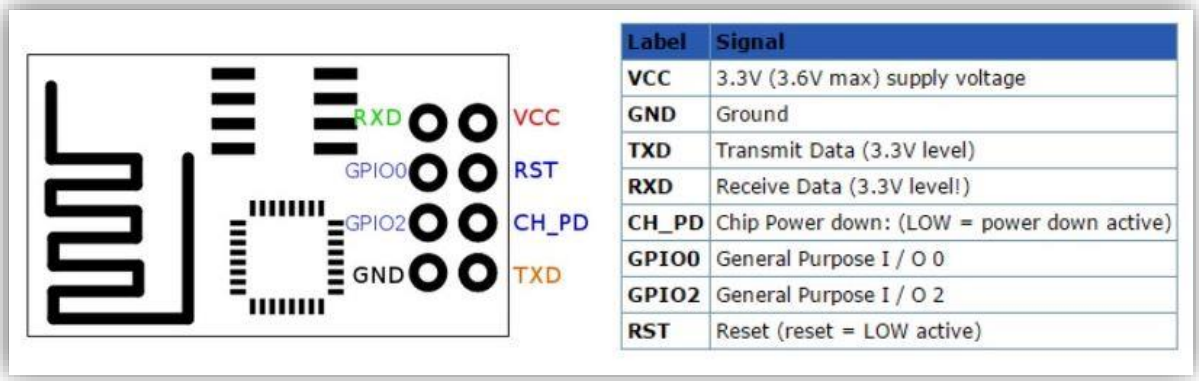


Figure 40. *ESP-01 (left) and USB/TTL Serial Adapter (right) with pin connections*

A wide variety of firmware flashing applications exist. For best results, the NodeMCU firmware programmer was utilised, as others proved inconsistent (nodemcu/nodemcu-flasher, 2019). The next step involved using the Arduino IDE to upload some sample code - though additional steps were involved for the IDE to recognise the module, covered in the project repo README file. A Blink example was uploaded and the LED on the ESP8266 started blinking successfully.



Label	Signal
VCC	3.3V (3.6V max) supply voltage
GND	Ground
TXD	Transmit Data (3.3V level)
RXD	Receive Data (3.3V level!)
CH_PD	Chip Power down: (LOW = power down active)
GPIO0	General Purpose I / O 0
GPIO2	General Purpose I / O 2
RST	Reset (reset = LOW active)

Figure 41. *ESP-01 pinout (hobbyist.co.nz)*

Work was balanced between applying code to the ESP-01 as well as a NodeMCU microcontroller [Fig.42]. NodeMCU is not required but was useful for testing code in a more streamlined manner, due to less time wasted rebuilding the circuit to upload code to ESP and then connecting it to Arduino again. The choice was made to integrate ESP-01 as it takes up less breadboard space and can run off the Arduino's 5V line with an adapter. Specs for the NodeMCU are available in Appendix B.4.

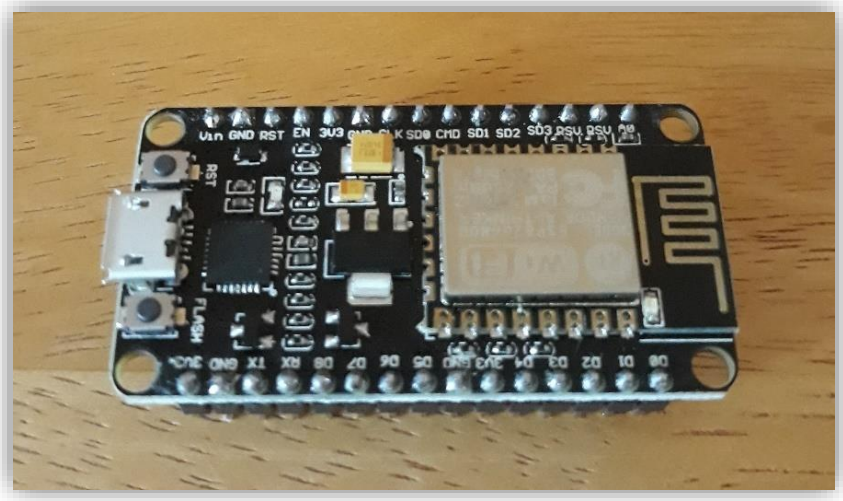


Figure 42. *NodeMCU IoT platform*

4.2.2 WiFi & OSC Configuration

The objective was always to send data from Arduino to ESP via serial communication, however several potential methods were considered and tested for relaying data from ESP to Pd over WiFi. These included concepts using the MQTT messaging protocol for IoT sensors, as well as more creative solutions through web server implementation and querying an SQL database logging sensor data. Ultimately, the best method was simply to introduce another OSC branch. This approach encouraged project extensibility, where code can be altered to scale data and send it directly from ESP to a DAW.

CSAW uses the main OSC library developed at CNMAT for Arduino integration (CNMAT/OSC, 2019). The GitHub repo contains a basic example for sending a /test OSC message from ESP to a server over a WiFi connection ("examples/ESP8266sendMessage") .

In this case, a laptop over a home WiFi network was the destination, so its IP was substituted along with the WiFi config settings into the example code. A simple Pd patch was created on the PC to listen for incoming requests on the port and print them to the console [Fig.43]. Once this was verified as working, integration with CSAW functionality followed - using this example as a model for Prototype B's code.

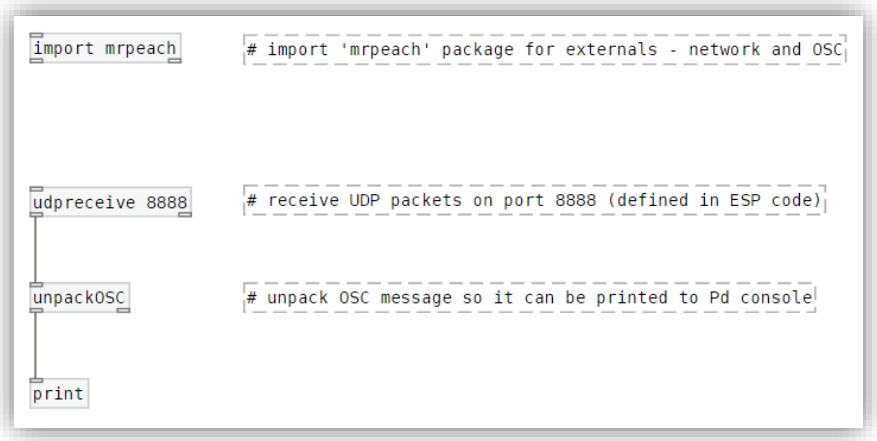


Figure 43. *Test Pd patch printing OSC messages from ESP8266*

4.2.3 CSAW-WiFi Integration

Extending CSAW's existing functionality began with adapting one of the potentiometer examples to send an OSC message containing the pot's value over the network to Pd. In code for the Arduino Mega, this required creating a new serial bus between Arduino and ESP through the SoftwareSerial library, included with Arduino's IDE by default (SoftwareSerial - arduino.cc, 2019). For ESP, this involved checking and reading the serial data, before converting it to an integer and adding it to an OSC message. Finally, in Pd, the same test patch was used to receive messages on the defined port - unpacking and printing to Pd's console.

Snippets of these functions with comments can be seen below, with the full code available in the project repo/USB. Once this was accomplished, the same logic was applied to other functions and sensors, sending the appropriate messages and data.

Serial_Arduino §

```
#include <SoftwareSerial.h>

// Create a serial bus communication between Arduino and ESP/NodeMCU, called 's'
SoftwareSerial s(10, 11);
// RX is pin 10, connect to TX of ESP/NodeMCU
// Tx is pin 11, connect to RX of ESP/NodeMCU

int pot1Val;

#define PIN1 A0

void setup(){
 s.begin(115200); // faster baudrate, ESP generally configured at 115200 by default
}

void loop()
{
  // read in the pot value, scale it between 1 and 100 and print it to the serial bus 's'
  pot1Val = analogRead(PIN1);
  pot1Val = map(pot1Val, 0, 1023, 1, 100);
  s.println(pot1Val);
  //Serial.println(pot1Val); // uncomment here to check data is printing to Arduino Serial Monitor
}
```

Figure 44. *"Serial_Arduino" sketch for Serial OSC example*

Serial_ESP01 §

```
void loop() {
  // if there is a value available from the Serial bus
  if (Serial.available()) {
      // Read in the serial data to var serialChar
      int serialChar = Serial.read();
      if (isDigit(serialChar)) {
        // saved incoming data is a byte so convert this to a char and add it to placeholder string
        placeString += (char)serialChar;
      }
      // if you get a newline (i.e for each new value)
      if (serialChar == '\n') {
        // convert the placeholder string, it's now an integer!
        val = placeString.toInt();

        // if statement checks that the value is not equal to the last value sent
        // if it is then the pot/value hasn't changed, so don't send another OSC message
        // this prevents OSC message flooding to Pure Data
        if (val != lastVal) {
          // Create a new OSC message under /test moniker
          // Add the integer value and perform UDP packet operations
          OSCMessage msg("/test");
          msg.add(val);
          Udp.beginPacket(outIp, outPort);
          msg.send(Udp);
          Udp.endPacket();
          msg.empty();
        }
        // Set the lastVal to current value for next loop cycle and reset the placeholder string
        lastVal = val;
        placeString = "";
      }
  }
}
```

Figure 45. *"Serial_ESP01" sketch for Serial OSC example*

Data read from serial needed to be converted to an integer, since the `Serial.read()` function automatically translates data as bytes. This would result in a byte stream where values increase/decrease as expected with pot movements but only up to 255 (the maximum decimal notation in bytes) before resetting back to 0 again. ESP code used the String to Int function - `toInt()` – to properly convert values.

The sketch also resulted in too many OSC messages being sent to Pd, causing an ECONNRESET error and force-closing the connection. Given that this was from a single potentiometer this was a concern, so two separate error-handling methods were undertaken. The first involved scaling pot data between 0 and 100 using the map function on Arduino's side, reducing the granularity of the data but also greatly decreasing the number of incoming messages due to less rapidly changing values. This still provided enough certainty to respond accurately to parameter changes.

The second method concerned value handling, whereby only integer values different from the previously sent value were sent as OSC messages. This was controlled with a simple variable check in an if statement, overwriting it at the end of every loop iteration. This still maintained a bit of jitter but resulted in substantially fewer messages coming through to Pd. This check was committed on the ESP side before creating OSC messages rather than Arduino to offload some processing.

The Serial_OSC WiFi sketches can be found in Appendix D.7.

4.2.4 Final B Prototype

Once this functionality was proved to work effectively, it was incorporated with a copy of the Prototype A folder. Some minor adjustments were required for the final Pd patch in order to detect OSC messages and pass just the numerical data through to the original patch framework.

The final prototype patch and sketches can be found in the project repository and USB under the "Prototype B" directory.

4.3 Wireless Desk Prototype – Prototype C

The building of the CSAW controller desk was outsourced to a local sign and display solutions company - SignSpec, given the final measurements diagram in Appendix C.8. The outcome was an aluminium surface board to the exact specifications of the diagram, along with some plastic legs attached via silicon for easier placement of the controller on a surface.

The final CSAW control surface for demonstration and testing (minus final jumper-wires) can be seen in Fig.46 below.

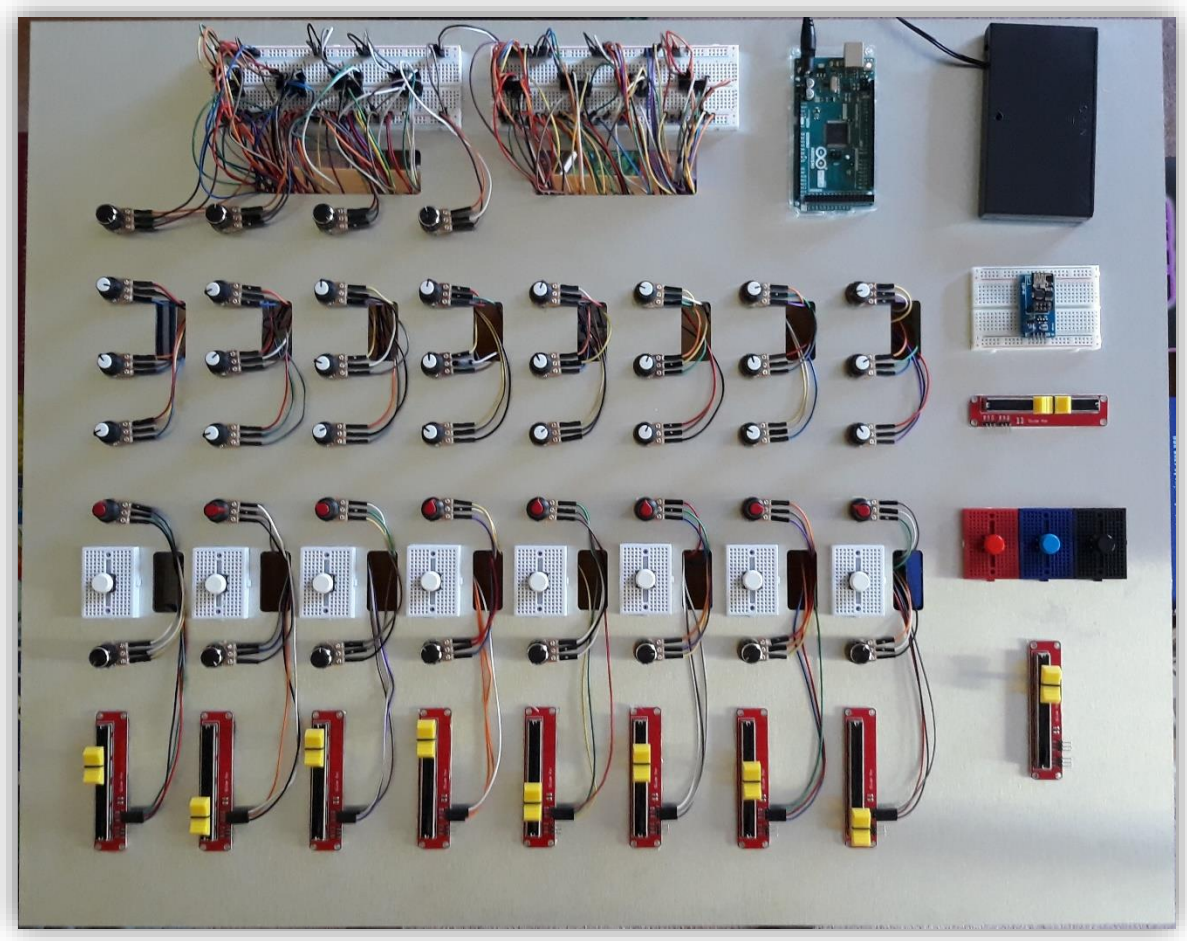


Figure 46. *CSAW final prototype*

4.4 Testing

4.4.1 Black-box testing

User testing for CSAW comprised of black-box testing methods; methods where the system's internal structure and design was not known to the tester (Differences between Black-box testing and White-box testing, 2019). None of the internal mechanics, processing or code was shown to test subjects outside of setup and connecting the controller. Afterwards, they were left to use it at their own convenience. Testers covered a range of ages with various backgrounds and interests in audio production (including some with no experience) and were provided with a questionnaire, before and after using the device (Appendix E).

The concluding part of the questionnaire consisted of 10 statements based on the System Usability Scale, a "quick and dirty" tool developed by John Brooke for measuring a system's usability (Brooke, 1986). Respondents were asked to rate each statement out of five responses from "Strongly agree" to "Strongly disagree". The scale has a complex scoring system, requiring normalization of data to produce a percentile ranking (above 68 is considered "above average"). Due to last-minute implementation requirements for the CSAW controller taking precedence close to the project deadline, the choice was made to perform further user testing at a later date with more testers and revisit and calculate SUS results.

4.4.2 White-box testing

White-box testing involves testing methods concerning the internal structure and design of a system and were largely carried out during the implementation stages of the project. Due to CSAW's iterative development model, testing was required as functionalities were integrated and revised regularly. Certain test cases were also carried out at various stages to ensure functional requirements were met continuously (refer to Section 3.1.2) - specifically analysing functionality by potentiometer and minimal latency communicating data at routine intervals.

5) Conclusions, Discussion & Future Work

5.1 Evaluation & Further Scope

Based on the project's development and testing results, some usability outcomes were discovered and noted. Many proved more beneficial than necessary, some of which have already been touched upon in the design and implementation sections. However, while none of them proved detrimental to the proof of concept, there are certainly areas that can be resolved or expanded upon based on CSAW's development process and user testing feedback.

5.1.1 Controller Evaluation

Transport controls

A creative choice to not bog down the device with a host of buttons, certain transport capabilities are missing from CSAW that many control surfaces possess - specifically rewind/fast-forward functions. This was a main outcome from user testing, some of whom found the play/stop mechanic cumbersome for resetting the test session. Based on the controller dimensions, with ample space for more pushbuttons, this could easily be integrated in the future.

Size/dimensions

Section 3 demonstrates the amount of attention put into designing the CSAW controller and its relative dimensions. Admittedly, the device size is bigger than most of this nature, and feedback confirmed this. This was a conservative choice based on the large number of components but, retrospectively, the size of the controller could probably be reduced by a third to a half.

Motors & Additional Channels

Many control surfaces utilise additional channel banks using motorized faders, another function that could be integrated with additional time and resources. In this manner, the controller would require stepper motors or, alternatively, motorized potentiometers. Since these would have been costly and would have increased the size of the desk even further, this was not considered for the finished product.

LCDs/LEDs

Another cosmetic addition, LCDs could be incorporated to detail exact pot values, for track numbers, etc. Similarly, LEDs would have proved beneficial for the likes of the solo pushbuttons, where some users found it problematic to determine when multiple tracks were soloed without looking at the DAW directly. While control surfaces are intended to be used in conjunction with DAWs and not as a replacement, it would be considerate to light an LED over each button when the track is soloed.

Stereo tracks

CSAW has eight channel strips based around the layout of the Behringer Xenyx X1222USB analog mixer. Unlike the Xenyx, which has 4 mono channels and 4 stereo channels, CSAW has 8 mono channels. Stereo control functionality could be achieved quite simply based on the dual analog pots used for the device, housing extra pins to split into left and right channels. A similar measure could be executed for the master fader via a neighbouring potentiometer.

5.1.2 Software Evaluation

MIDI

OSC was chosen as a communication protocol over MIDI despite most control surfaces using MIDI for function mapping. Despite OSC's capability of providing higher precision levels among other benefits (see Section 2.3.3), it is a little limiting as MIDI is the more pervasive

option in this field. One of the next phases of CSAW would be to coordinate OSC and MIDI support, providing further choice for DAWs, many of which do not currently have OSC support.

Pure Data requirement

Section 4 details the necessity of Pd for this project, based on OSC libraries available to Arduino (see **A note on Serial Data, libraries & Pure Data**). A potential next step for CSAW could be to cut out Pd completely. However, as previously specified, it was extremely beneficial to this project, aiding testing and troubleshooting during the implementation stage, and providing a more manageable solution for processing sensor data.

Programmable parameters

CSAW was developed to replicate primary analog desk functionality, however, many control surfaces have easily programmable/assignable rotary encoders or buttons for wider variety of DAW functions. This idea was considered during the project timeline, however the level of software development required to achieve this in the allotted time was not feasible. This could be expanded efficiently through Pd, designing a graphical user interface which interacts with DAW APIs – though the time required to map a host of plugins/actions, and define conditional logic for them would be excessive.

Standalone software

Taking this approach even further, programming an executable application or plugin to work with a DAW would be an effective step for taking the project forward – particularly for initial setup. Programming languages like C++ provide an additional level of flexibility for application interaction. This would also be more beneficial for configuring CSAW's network functions which currently need to be coded and uploaded to Arduino and ESP.

5.2 Conclusion

In conclusion, the CSAW audio control surface successfully amounts to the principal capabilities of a DAW controller, proving itself among the many software-centric systems that currently dominate the market. With its recognisable yet intuitive design, sophisticated software foundation and built-in WiFi transceiver module for consistent wireless facilities, CSAW exhibits an ergonomic digital workstation controller for the modern audio engineer, providing a key functionality that is ignored by many wholesale controllers. Through the use of exclusively fundamental electronics and strictly open-source technologies and platforms, CSAW proves competent in matching many of the commercial physical products available in the public domain at this time.

Appendices

Appendix A: Journal Paper

CSAW: An audio **C**ontrol **S**urface built using the **A**rduino platform and **W**ireless capabilities

Fiach O'Donnell, C.I.T Cork School of Music

fiach.odonnell@mycit.ie

***Abstract*—Audio control surfaces are practical tools for editing and mixing in a digital audio workstation. Despite recent technological advancements, however, there is a lack of robust physical commercial controllers with seamless wireless functionality. This paper describes the design and implementation of CSAW – an audio Control Surface built using the Arduino platform and Wireless capabilities. CSAW aims to remedy limitations of mobile control applications and certain hardware controllers through devising a proof of concept control surface and software platform using sophisticated open-source applications and wireless communication protocols.**

***Index Terms*—Arduino, control surface, OSC, Pd, ESP8266, wireless, DAW**

NOMENCLATURE

CSAW	Control Surface with Arduino Wireless capabilities
DAW	Digital Audio Workstation
ESP	ESP-01 ESP8266 WiFi Module
Pd	Pure Data
OSC	Open Sound Control

I. INTRODUCTION

THE advent of smart phone technology and software applications in the modern age has undoubtably seen an increased interest in wireless and remote audio control within the audio production spectrum. The consolidation of music with the Internet of Things (IoT) in recent years has surpassed the expectations of engineers and enthusiasts alike, who continue to discover and implement new means of audio manipulation at a rapid rate.

Despite these advancements, however, there is still quite a lack of robust wireless capabilities for physical controllers/control surfaces on the market. Big name brands such as Behringer influence the pack with products like the X-Touch, but even so, this type of control is not as widely implemented as expected, considering the universal adoption of WiFi in the late 90s. For audio professionals, the necessity of a physical controller is imperative, given the frequent need to make multiple changes simultaneously, in contrast to the visual limitations and un-natural feel of changing parameters on the small screen.

The CSAW project aims to remedy this through utilising and consolidating existing technologies and hardware to construct a functional wireless physical control surface for audio manipulation. Through the use of sophisticated applications and communication protocols, along with a fundamental electronics environment with elemental components, a proof of concept controller and software platform will be devised to enable the audio engineer to perform mixing and recording functions in an easier, more efficient manner than its software counterparts.

F. O'Donnell is with C.I.T Cork School of Music, Union Quay, Ballintemple, Cork, Ireland (e-mail: fiach.odonnell@mycit.ie).

II. BACKGROUND & DESIGN

An audio control surface refers to a device or interface that "allows the user to control a digital audio workstation or other digital audio application" [1]. These range from the primitive, such as Behringer's X-Touch Mini [2] with its single fader and 8 rotary potentiometers for track volume control, to the more elaborate, like Avid's Pro Tools S3 [3], which allows for a wider range of functionality such as automation, routing management, etc. CSAW acts as such an interface, allowing for basic DAW and track control implemented through Arduino and WiFi capabilities.

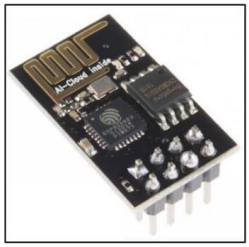


Fig. 1. ESP-01: ESP8266 WiFi module

At its core, CSAW design is closer to a combination of control surface and analog console than to a typical control surface. The controller includes analog sensors with pre-mapped controls for the more common DAW functions. It also makes use of DAW built-in OSC capabilities instead of sluggish WiFi setup and configuration over some of its counterparts.

CSAW houses a WiFi transceiver, the ESP-01 (Fig.1) [4], a model of the ESP8266 module to provide easier connectivity for the core Arduino component to a pre-existing network. Making use of simple, affordable electronic potentiometers and push buttons that connect to the Arduino Mega [5], the

CSAW control surface keeps to a compact design. Layout is based on the Behringer Xenyx X1222USB analog mixer [6].

The Pure Data [7] visual programming language is utilised as middleware software, collecting sensor messages from the controller Arduino component and processing and broadcasting them to a DAW. These messages are communicated via Open Sound Control [8] – a message-based protocol for communication among computers, sound synthesizers and other multimedia devices and applications that is optimized for network distributed music systems. OSC was selected over the likes of MIDI based on its user-defined message structure and robust ability to define different data types among others.

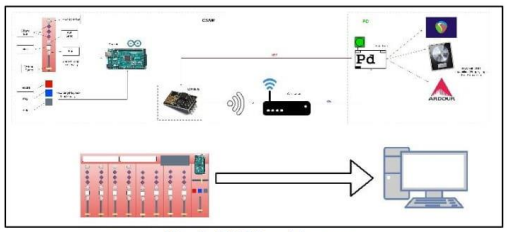


Fig. 2. CSAW architecture

III. IMPLEMENTATION

Two separate communication methods were used to send audio control to a DAW on a computer; a) transmitting control signals via USB (wired connection) or b) via WiFi using the ESP8266 chip (Fig.2). The Arduino Mega component scaled analog/digital data read from the controller's potentiometers and pushbuttons before sending them via serial to either USB or ESP. The numerical data was then received on a local instance of Pd running on the DAW PC through either method - the latter utilising OSC for ESP->Pd transmission. Both methods then utilised Pd to configure separate OSC messages to map data to built-in DAW actions, performing the appropriate task (e.g. volume control, equalization, etc.).

All potentiometers and resistors used were of 10kΩ resistance – the optimum value as anything smaller consumes current while anything larger takes longer to read or can be affected by values of other pots. Multiplexers were employed to extend analog inputs due to limitations of Arduino. The project applied 8xCD4051BE 8-channel muxes, developed by Texas Instruments Inc., to route sensor inputs for each channel strip to Arduino's analog pins.

CSAW makes use of contributions from the GitHub community for streamlining controller operations. The two primary repositories that were invoked during implementation were 'Arduino_Pd' – a library of Pd abstractions for facilitating reading of serial data [9] – and the main OSC library developed at CNMAT for Arduino integration [10].

Desk construction was outsourced to a local sign and display solutions company, given a measurements diagram (Fig.3). Basic black-box user testing comprised querying statements to testers based on John Brooke's System Usability Scale [11], for measuring the system's usability.

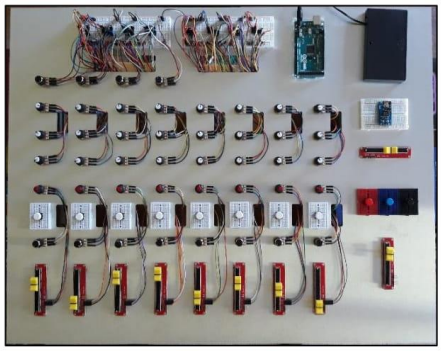


Fig. 3. CSAW final prototype

IV. CONCLUSION

The CSAW audio control surface successfully amounts to the principal capabilities of a DAW controller, proving itself among the many software-centric systems that currently dominate the market. With its recognisable design and built-in WiFi transceiver for consistent wireless facilities, CSAW exhibits an ergonomic digital workstation controller for the modern audio engineer, providing a key functionality that's ignored by many wholesale controllers.

Through its use of exclusively fundamental electronics and strictly open-source technologies and platforms, CSAW proves competent in matching many of the commercial physical products available in the public domain at present.

V. ACKNOWLEDGEMENTS

This work was presented as part of the requirements for the MSc. in Music & Technology, awarded by C.I.T Cork School of Music (CSM). The author would like to thank Hugh McCarthy & Paddy Collins among other CSM lecturers, along with course colleagues and the author's parents.

VI. REFERENCES

[1] Wikipedia contributors. Audio control surface. In Wikipedia, The Free Encyclopedia. Retrieved 12:01, August 12, 2019, from https://en.wikipedia.org/w/index.php?title=Audio_control_surface&oldid=827929386
[2] Behringer. (2014). X-TOUCH MINI: Quick Start Guide.
[3] Avid. (2014). Pro Tools | S3 User Guide.
[4] AI-Thinker. (2015). ESP-01 WiFi Module Version 1.0.
[5] Arduino Mega. (2019) arduino.cc: https://store.arduino.cc/mega-2560-r3
[6] Behringer. (2019). Xenyx X1222USB User Manual.
[7] Puckette, M. S. (2016). Pure Data.
[8] Wright, M., Freed, A., & CNMAT. (1997). Open Sound Control: A New Protocol for Communicating with Sound Synthesizers.
[9] alexdrymonitis/Arduino_Pd. Retrieved from GitHub.com: https://github.com/alexdrymonitis/Arduino_Pd
[10] CNMAT/OSC. Retrieved from GitHub.com: https://github.com/CNMAT/OSC
[11] Brooke, J. (1986). SUS - A quick and dirty usability scale.

Appendix B: Technical Specifications

Appendix B.1: Arduino Mega 2560 Rev3

Microcontroller	Atmega2560
Operating Voltage	5V
Input Voltage	7V – 12V
USB Port	Yes
DC Power Jack	Yes
Current Rating Per I/O Pin	20mA
Current Drawn from Chip	50mA
Digital I/O Pins	54
PWM	15
Analog Pins (Can be used as Digital Pins)	16 (Out of Digital I/O Pins)
Flash Memory	256KB
SRAM	8KB
EEPROM	4KB
Crystal Oscillator	16 MHz
LED	Yes/Attached with Digital Pin 13
Wi-Fi	No
Shield Compatibility	Yes

Arduino Mega 2560 Specifications

www.TheEngineeringProjects.com

Source: *theengineeringprojects.com*

Appendix B.2: ESP-01

Categories	Items	Values
WiFi Paramters	WiFi Protocles	802.11 b/g/n
	Frequency Range	2.4GHz-2.5GHz (2400M-2483.5M)
Hardware Paramaters	Peripheral Bus	UART/HSPI/I2C/I2S/Ir Remote Contorl
		GPIO/PWM
	Operating Voltage	3.0~3.6V
	Operating Current	Average value: 80mA
	Operating Temperature Range	-40°~125°
	Ambient Temperature Range	Normal temperature
	Package Size	14.3mm*24.8mm*3mm
	External Interface	N/A
Software Parameters	Wi-Fi mode	station/softAP/SoftAP+station
	Security	WPA/WPA2
	Encryption	WEP/TKIP/AES
	Firmware Upgrade	UART Download / OTA (via network) / download and write firmware via host
	Ssoftware Development	Supports Cloud Server Development / SDK for custom firmware development
	Network Protocols	IPv4, TCP/UDP/HTTP/FTP
	User Configuration	AT Instruction Set, Cloud Server, Android/iOS App

Source: *ESP-01 WiFi Module Version1.0 (AI-Thinker, 2015)*

Appendix B.3: CD4051BE & 74HC4067 features

- Wide Range of Digital and Analog Signal Levels
  - Digital: 3 V to 20 V
  - Analog: ≤20 V_{P-P}
- Low ON Resistance,125 Ω (Typical) Over 15 V_{P-P} Signal Input Range for $V_{DD} - V_{EE}$ = 18 V
- High OFF Resistance, Channel Leakage of ±100 pA (Typical) at $V_{DD} - V_{EE}$ = 18 V
- Logic-Level Conversion for Digital Addressing Signals of 3 V to 20 V ($V_{DD} - V_{SS}$ = 3 V to 20 V) to Switch Analog Signals to 20 V_{P-P} ($V_{DD} - V_{EE}$ = 20 V) Matched Switch Characteristics, r_{ON} = 5 Ω (Typical) for $V_{DD} - V_{EE}$ = 15 V Very Low Quiescent Power Dissipation Under All Digital-Control Input and Supply Conditions, 0.2 µW (Typical) at $V_{DD} - V_{SS} = V_{DD} - V_{EE}$ = 10 V
- Binary Address Decoding on Chip
- 5 V, 10 V, and 15 V Parametric Ratings
- 100% Tested for Quiescent Current at 20 V
- Maximum Input Current of 1 µA at 18 V Over Full Package Temperature Range, 100 nA at 18 V and 25°C
- Break-Before-Make Switching Eliminates Channel Overlap

Source: *CD405xB datasheet (Texas Instruments, 2017)*

- Input levels S0, S1, S2, S3 and $\overline{E}$ inputs:
  - For 74HC4067: CMOS level
  - For 74HCT4067: TTL level
- Low ON resistance:
  - 80 Ω (typical) at V_{CC} = 4.5 V
  - 70 Ω (typical) at V_{CC} = 6.0 V
  - 60 Ω (typical) at V_{CC} = 9.0 V
- Specified in compliance with JEDEC standard no. 7A
- ESD protection:
  - HBM JESD22-A114F exceeds 2000 V
  - MM JESD22-A115-A exceeds 200 V
  - CDM JESD22-C101E exceeds 1000 V
- Multiple package options
- Specified from –40 °C to +85 °C and –40 °C to +125 °C
- Typical 'break before make' built-in

Source: *74HC4067 datasheet (nexperia, 2015)*

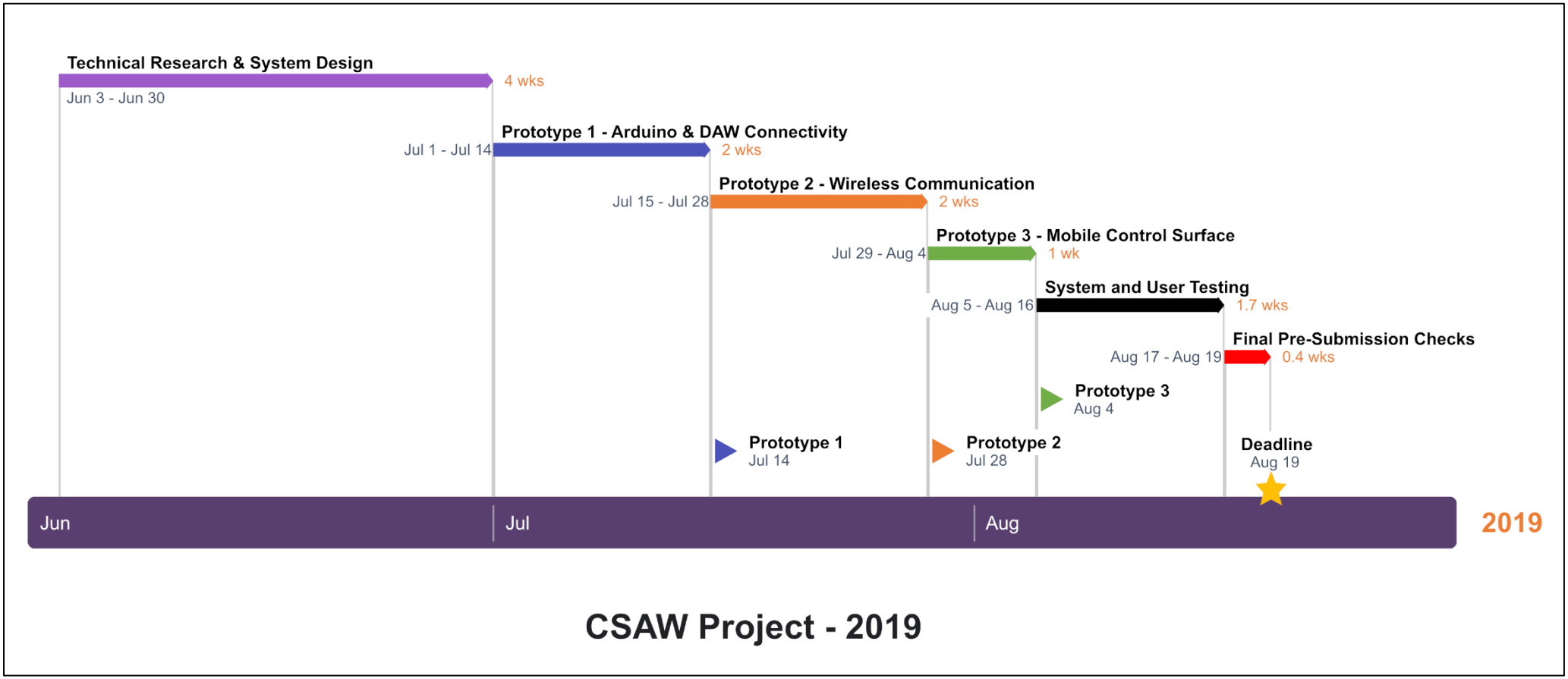
Appendix B.4 NodeMCU

Wi-Fi Protocols	802.11 b/g/n
Frequency Range	2400Mhz-2483.5Mhz
Tx Power	802.11 b +20 dBm
Rx Sensivity	-91 dBm (11 Mbps)
Peripheral Bus	UART/SDIO/SPI/I2C/IR Remote Control
Operating Voltage	3.0 - 3.6V
Operating Current	80mA (Average)
Operating Temperature Range	-40°C - 125°C
Wi-Fi Mode	Station/SoftAP/Soft AP+station
Security	WPA/WPA2
Encryption	WEP/TKIP/AES
Network Protocols	IPv4, TCP/UDP/HTTP/FTP
Microcontroller	Tensilica L106 32 bit
CPU Clock Speed	80Mhz
RAM Size	36KB
Standby Current	0.9 mA
Deep Sleep	10 uA
Power Save Mode DTIM1	1.2 mA
Power Save Mode DTIM3	0.86 mA

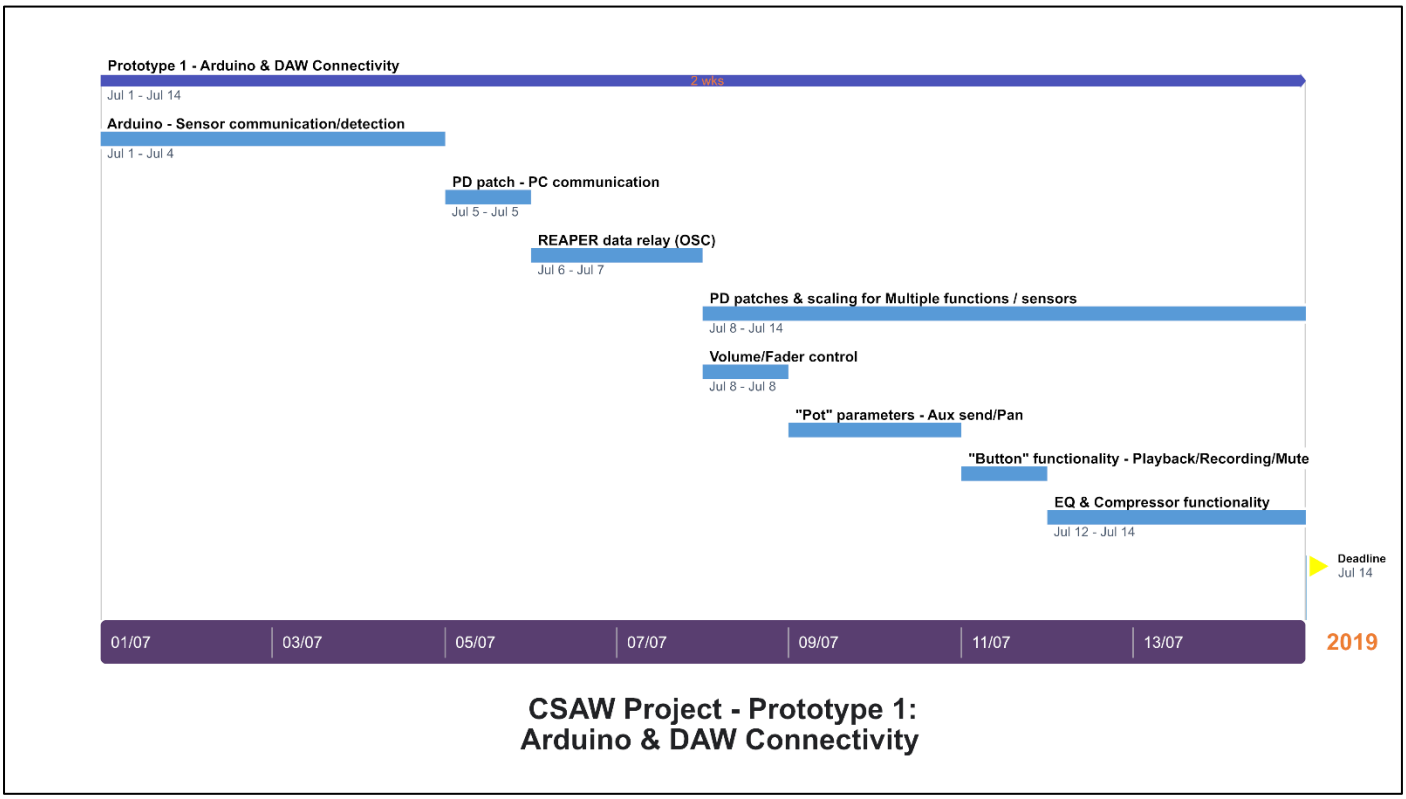
Source: *Designing a web-based data acquisition system for battery-powered wireless sensor nodes: WiFiLab (Kirbas, Ismail & Dukkanci, Ayhan, ICENS, 2018)*

Appendix C: Design Diagrams & Documents

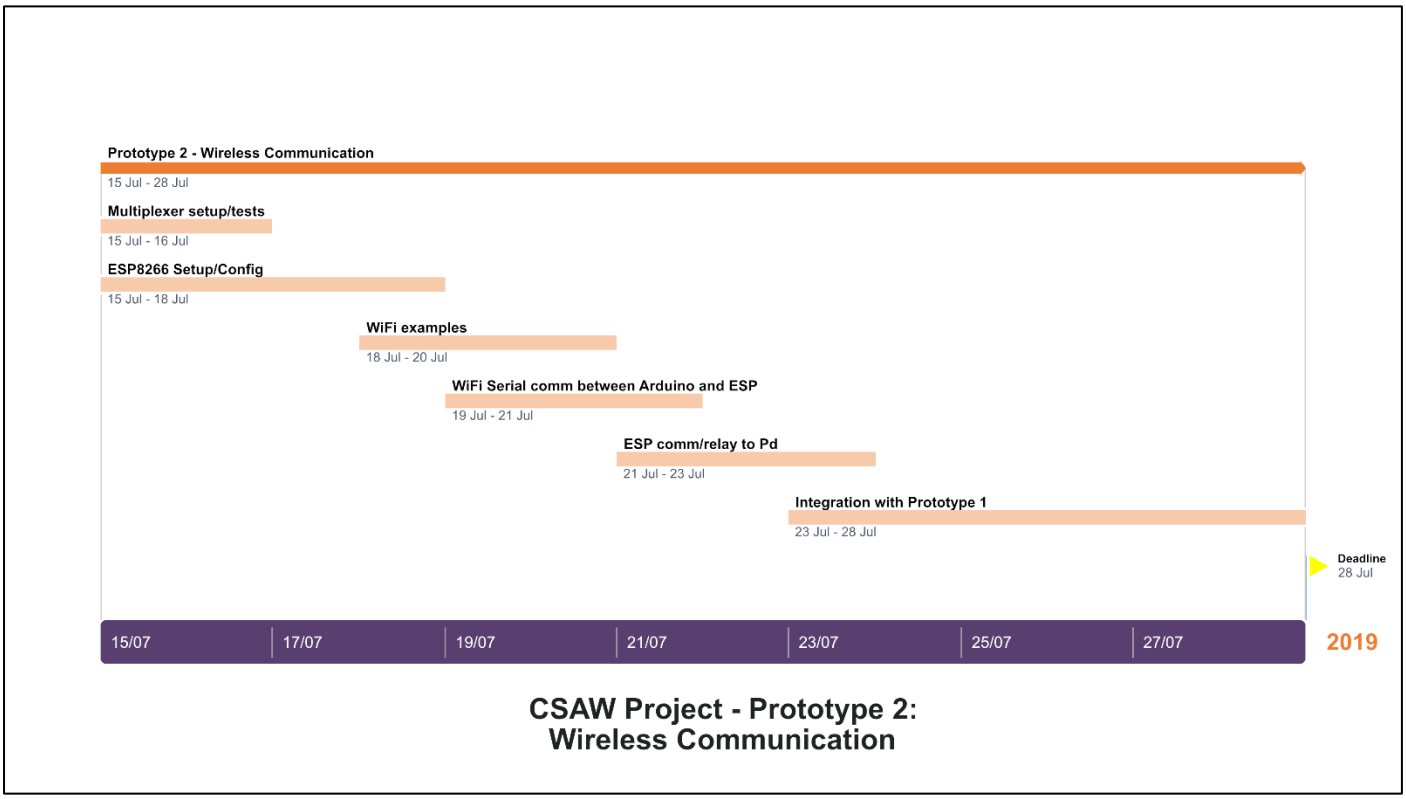
Appendix C.1: Gantt Charts



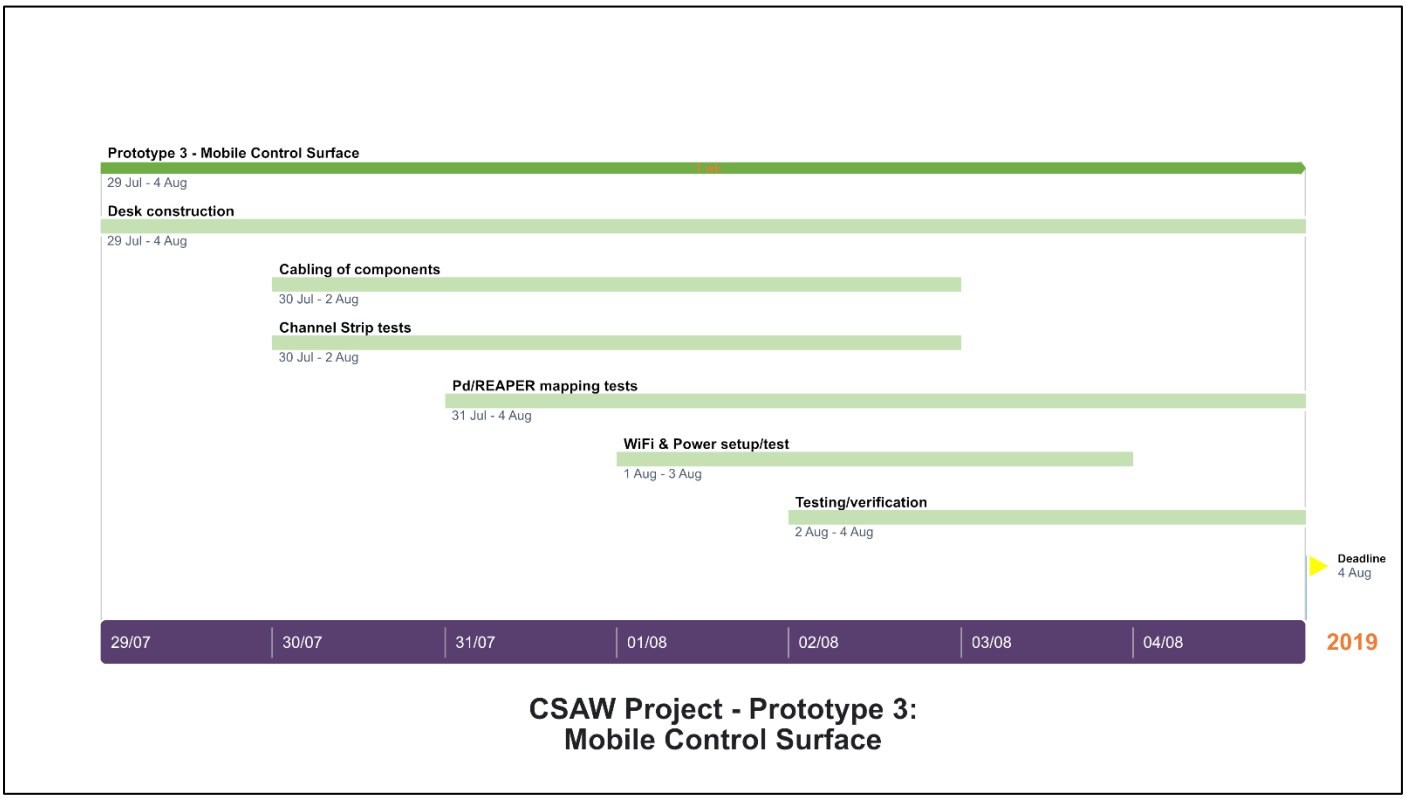
Main project Gantt Chart



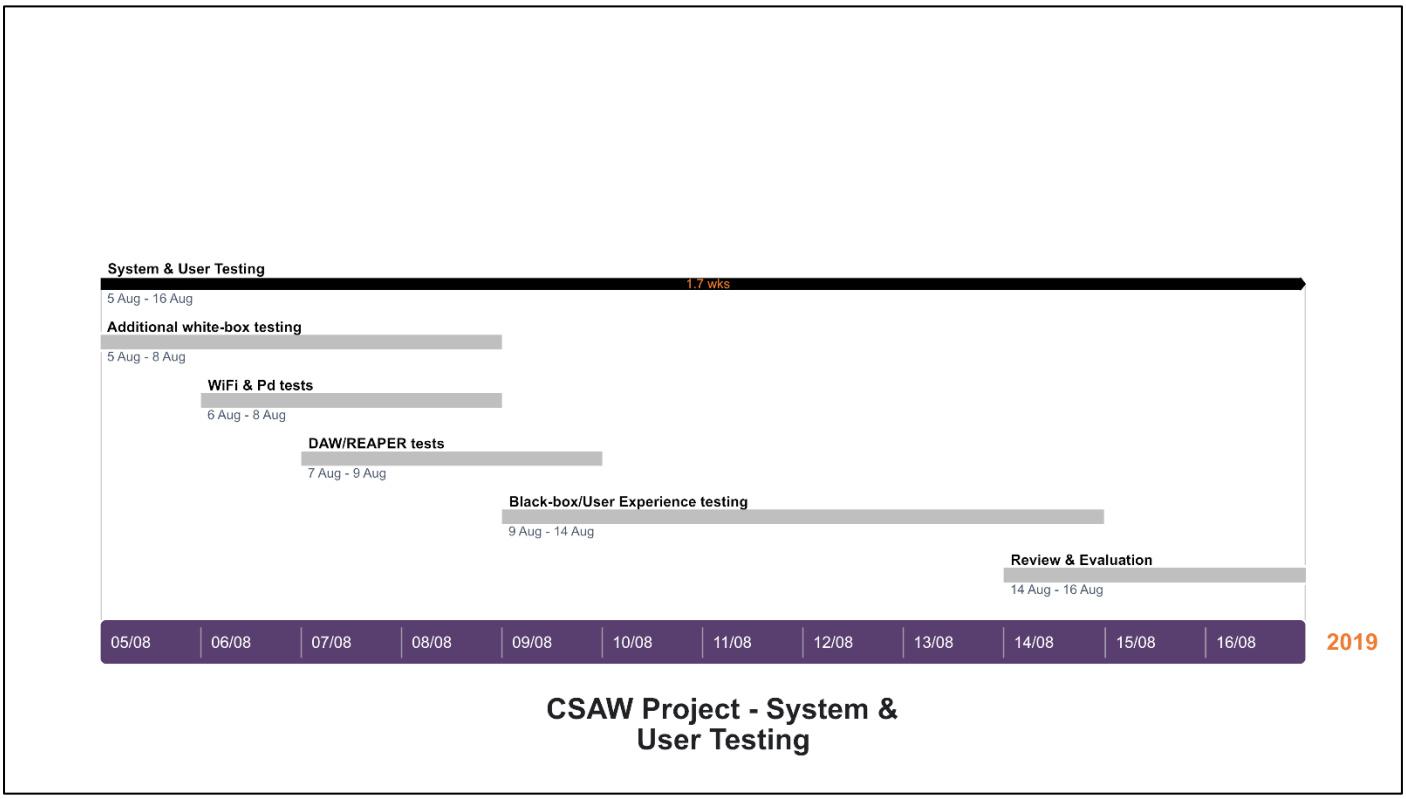
Prototype 1 Gantt Chart



Prototype 2 Gantt Chart

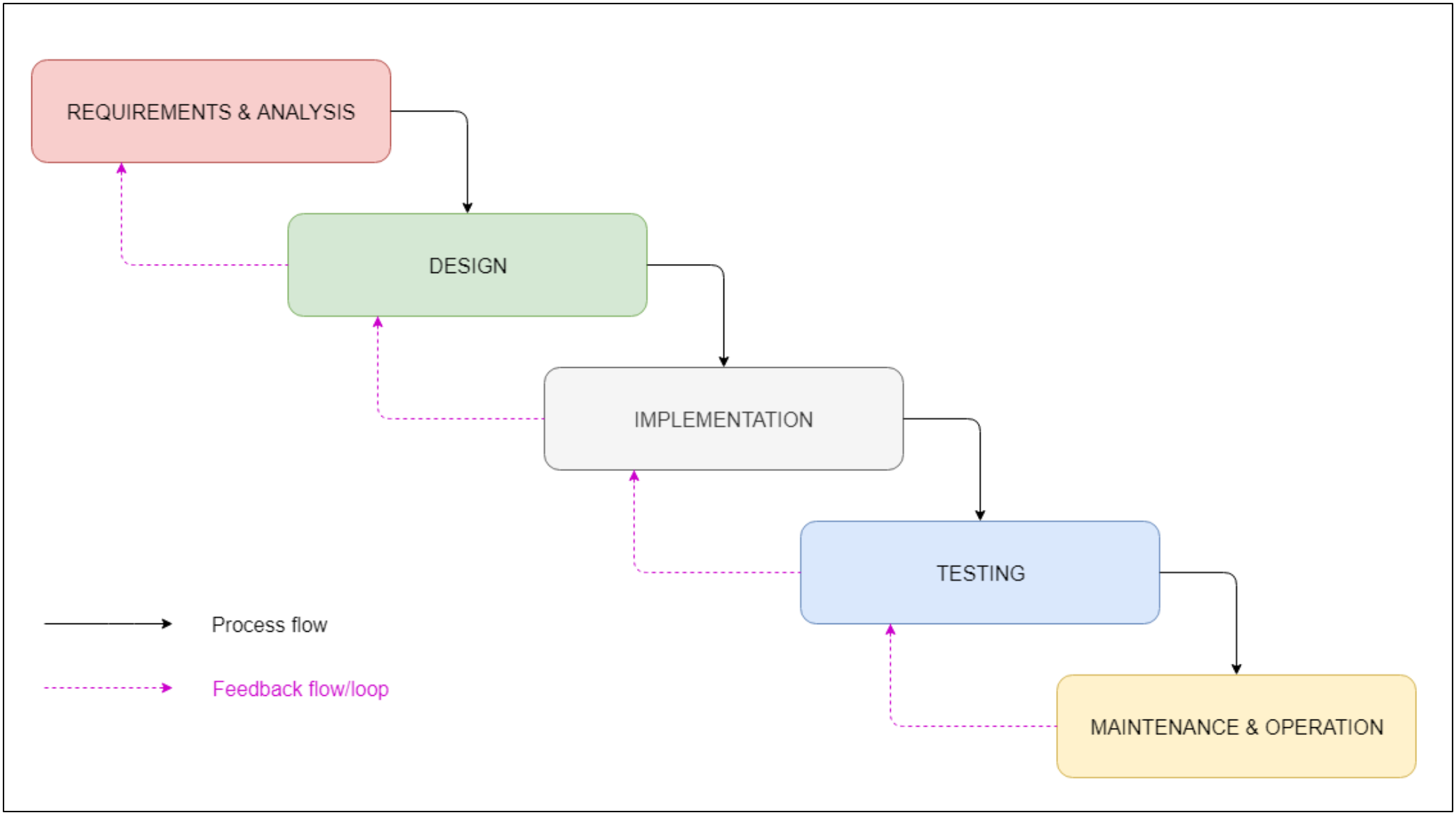


Prototype 3 Gantt Chart



Testing Gantt Chart

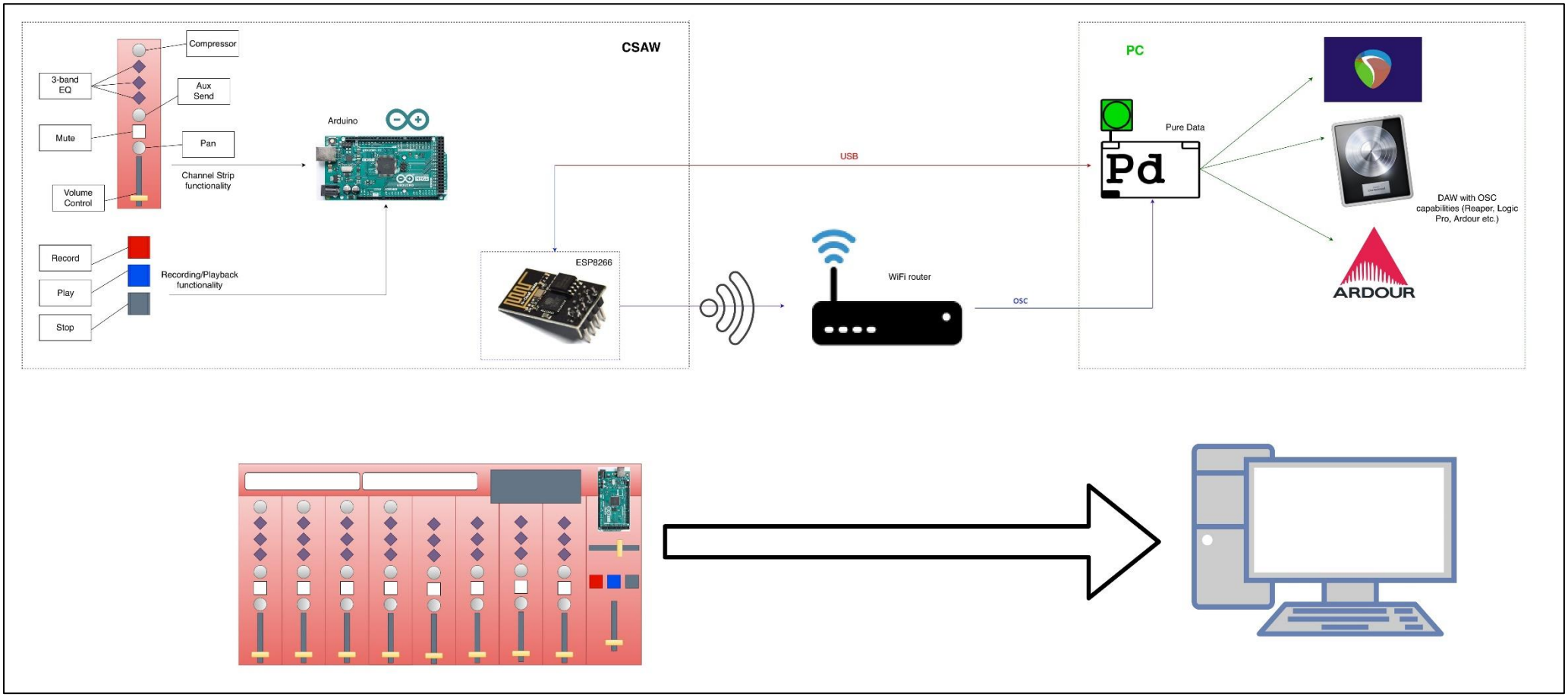
Appendix C.2 Waterfall SDLC Model



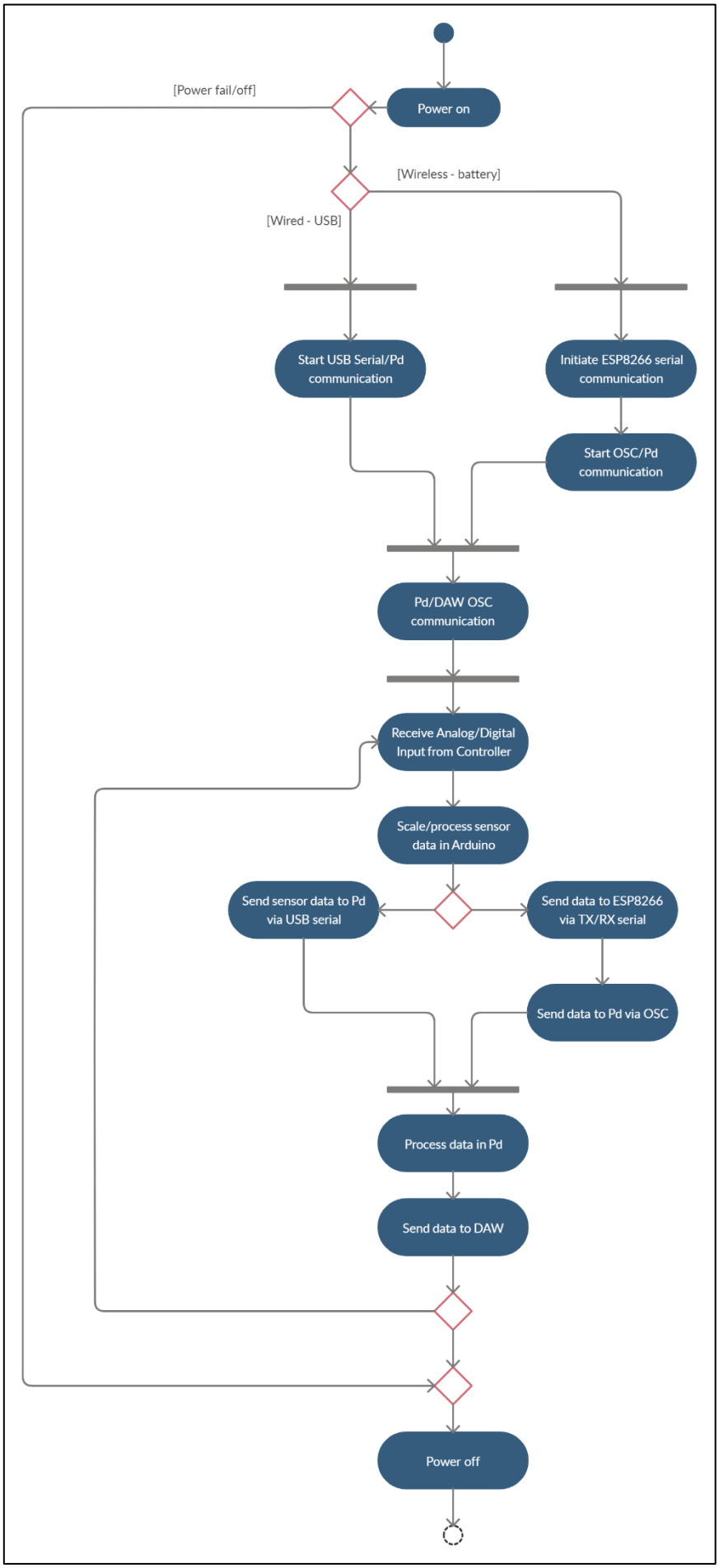
Appendix C.3 List of Materials

				Items in red receommended, may not be required		
	Component	**Quantity**	**Cost per unit**	**Comments**		
Arduino	Arduino Mega Rev3	1	€35.00	Purchased on arduino.cc - Arduino component for control surface	€35.00	
	Arduino Workshop Kit	1	€35.00	Purchased on arduino.cc - includes mini breadboard, Arduino USB cable, additional small jumper wires & more	€35.00	€70.00
Potentiometers & Buttons	10k Slide Potentiometer set (5pcs)	3	€9.72	Purchased on banggood.com - Double Analog Potentiometer Module With Slide Potentiometer	€29.16	
	YINNETECH 10k Rotary Potentiometer kit (20pcs)	1	€14.09	Purchased on amazon.co.uk (vendor: E-NETTECH)	€14.09	
	Wimas 10k Rotary Potentiometer kit 45pcs	1	€18.99	Purchased on amazon.co.uk (vendor: WiMas)	€18.99	
	Push Button set (50pk)	1	€4.34	Purchased on amazon.co.uk (vendor: YoumileDirect)	€4.34	€66.58
Jumper Wires	Jumper Wires set 30cm (150pcs)	2	€7.59	Purchased on amazon.co.uk (vendor: MakerStack)	€15.18	
	Jumper Wires set 20cm (120pcs)	2	€5.37	Purchased on amazon.co.uk (vendor: GanvolTech)	€10.74	
	Jumper Wires set 10cm (2 x 150pcs)	1	€6.50	Purchased on amazon.co.uk (vendor: MakerStack)	€6.50	€32.42
Breadboards	ELEGOO breadboard kit 830pt (3pcs)	1	€9.77	Purchased on amazon.co.uk (vendor: GYE UK)	€9.77	
	ELEGOO breadboard kit 170pt (6pcs)	1	€6.50	Purchased on amazon.co.uk (vendor: GYE UK)	€6.50	
	Electrely breadboard kit 170pcs (6pcs)	2	€6.16	Purchased on amazon.co.uk (vendor: Electrely)	€12.32	€28.59
Multiplexers	CD4051BE 8 channel mux (10pcs)	2	€6.04	Purchased on amazon.co.uk (vendor: V&U Electronic Components Ltd)	€12.08	€12.08
Wireless	Laqiya FT232RL USB TTL Converter (2pcs)	1	€9.23	Purchased on amazon.co.uk (vendor: Laqiya)	€9.23	
	ESP-01 WiFi Module with breadboard adapter	1	€7.05	Purchased on amazon.co.uk (vendor: AZ-Delivery-Shop)	€7.05	
	diymore ESP-01 Adapter Module	1	€6.51	Purchased on amazon.co.uk (vendor: diymore)	€6.51	
	USB Mini B cable	1	€1.73	Purchased on ebay.ie (vendor: bluechargedirect)	€1.73	
	ESP8266 NodeMCU dev board (2pcs)	1	€11.38	Purchased on amazon.co.uk (vendor: amazon.co.uk EU S.a.r.L)	€11.38	€35.90
Power & Misc.	MB102 Breadboard power supply adapter	2	€3.79	Purchased on amazon.co.uk (vendor: Hengjiaan) - Haljia model	€7.58	
	12V Battery Holder case	1	€6.51	Purchased on amazon.co.uk (vendor: DIGISLIDER) - 8 x AA holder	€6.51	
	12V power supply	1	€6.51	Purchased on amazon.co.uk (vendor: LightingWill-UK)	€6.51	
	Ultrics Digital LCD Multimeter	1	€10.09	Purchased on amazon.co.uk (vendor: ETHER UK)	€10.09	
	Resistor Assortment Kit (1280pcs)	1	€9.77	Purchased on amazon.co.uk (vendor: Aussel)	€9.77	
	Aluminium Desk surface	1	N/A	Quote required, obtained for free	N/A	€40.46
						€286.03

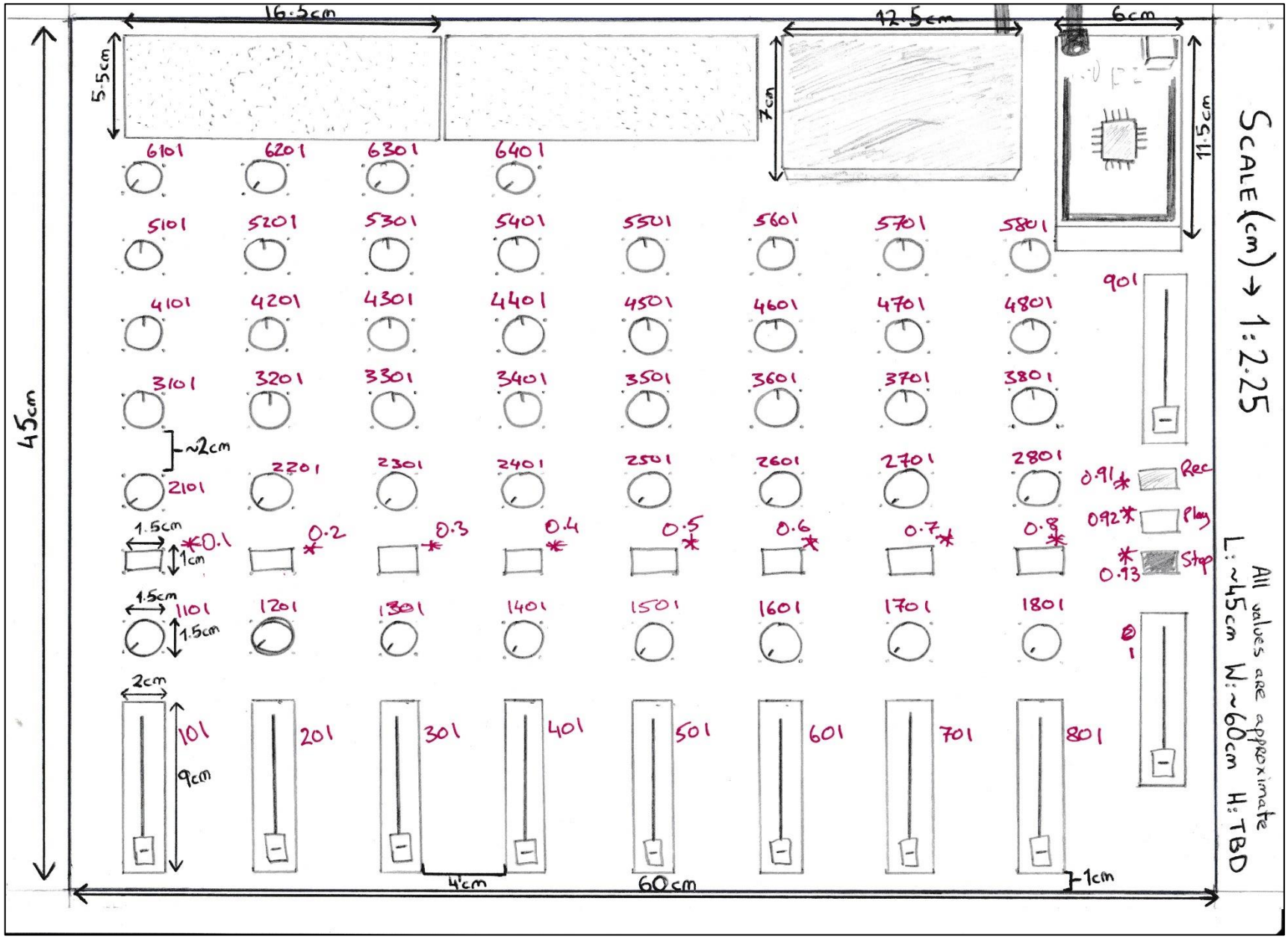
Appendix C.4 Architecture Diagram



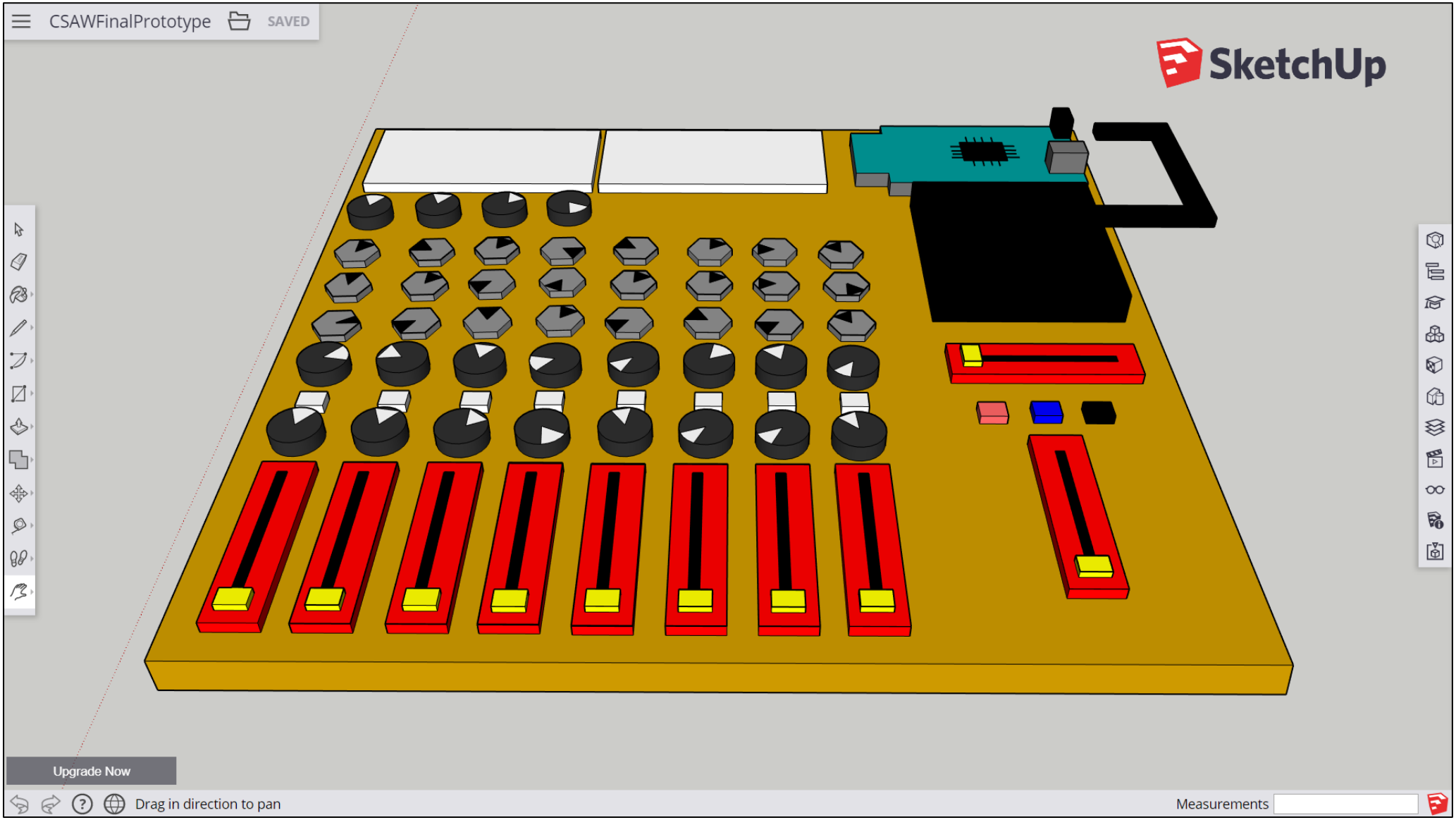
Appendix C.5 Activity Diagram



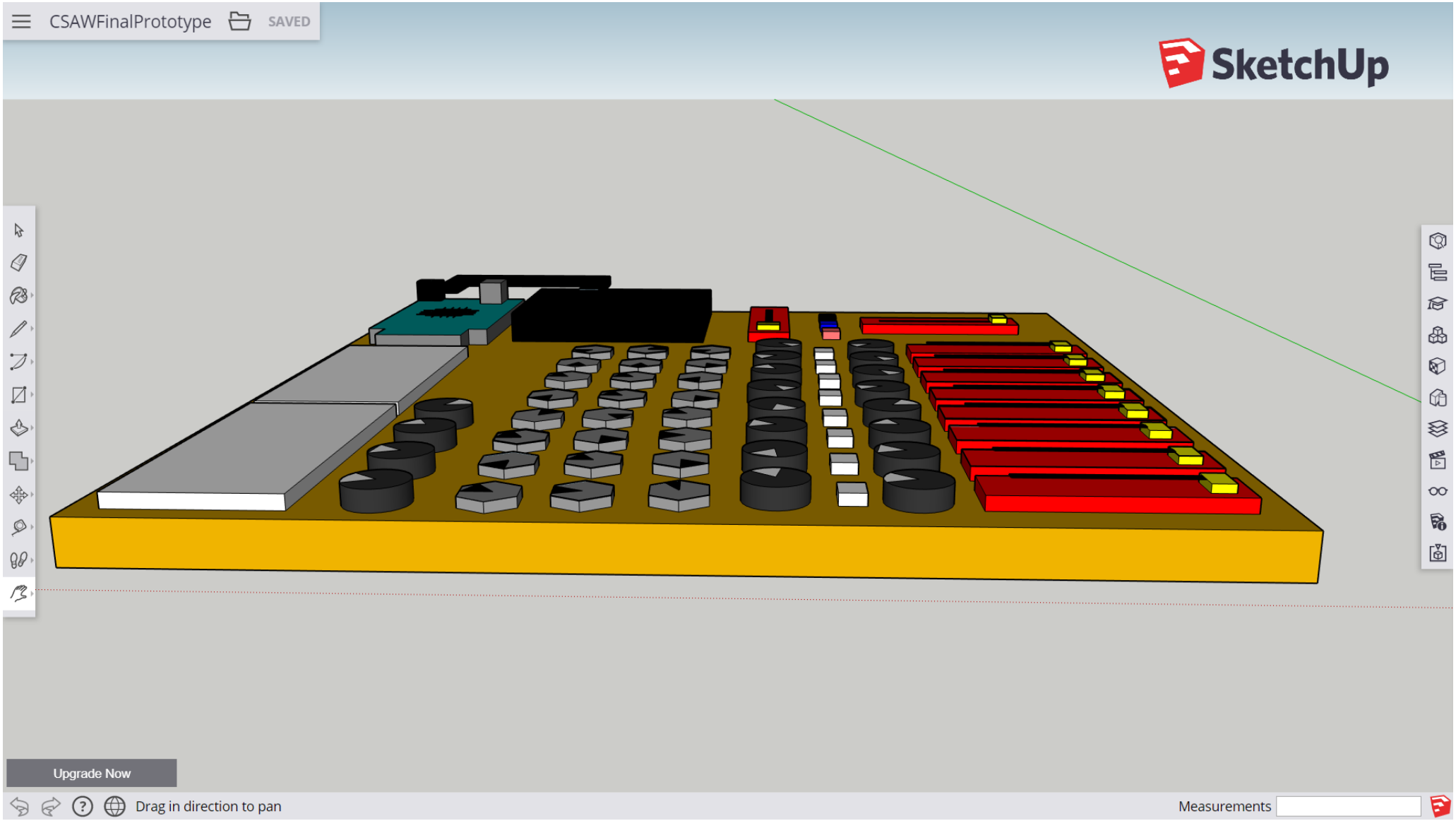
Appendix C.6 Scale Diagram/Numbering Scheme



Appendix C.7 3D Model Representation

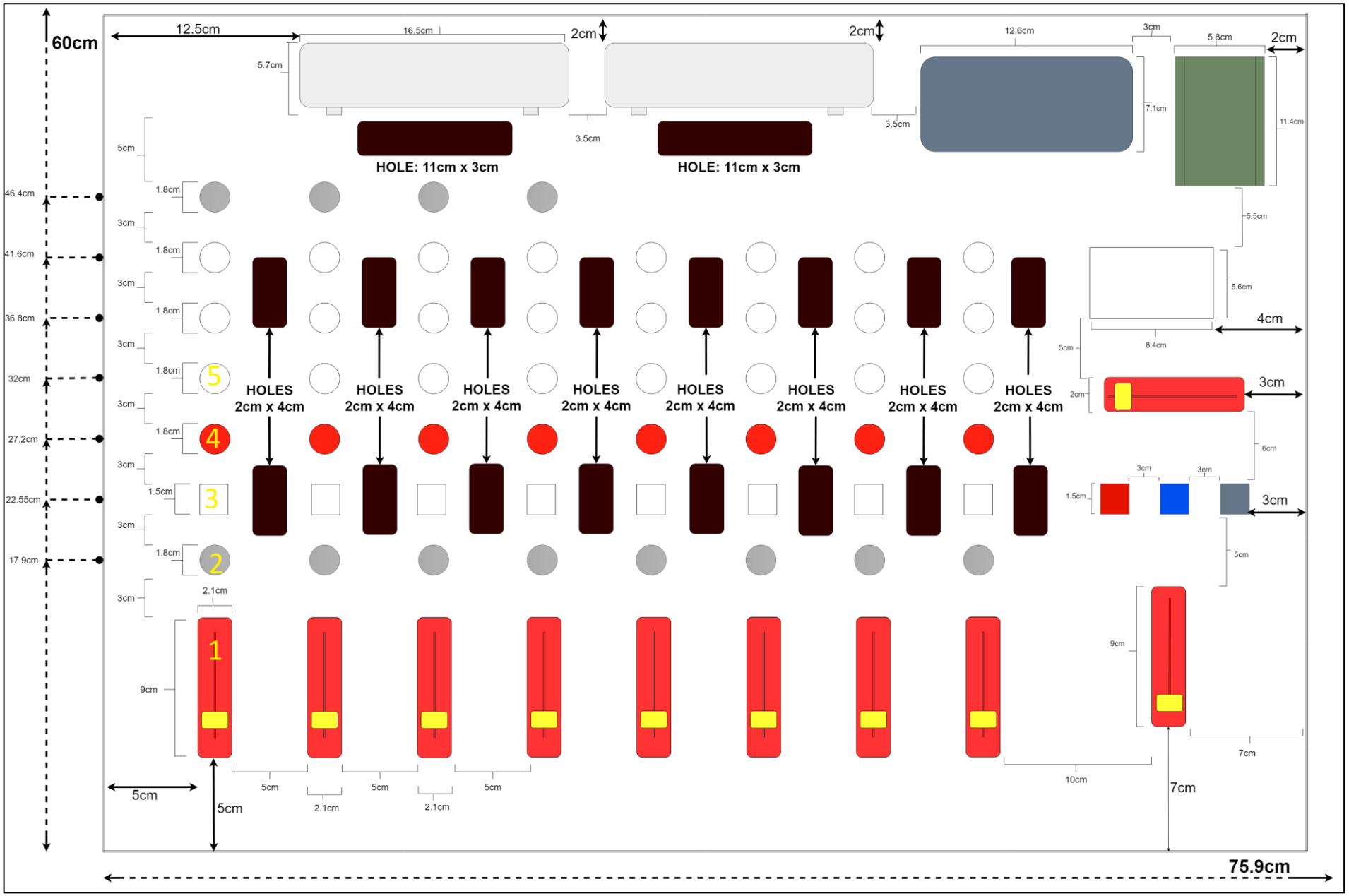


CSAW SketchUp – Top view

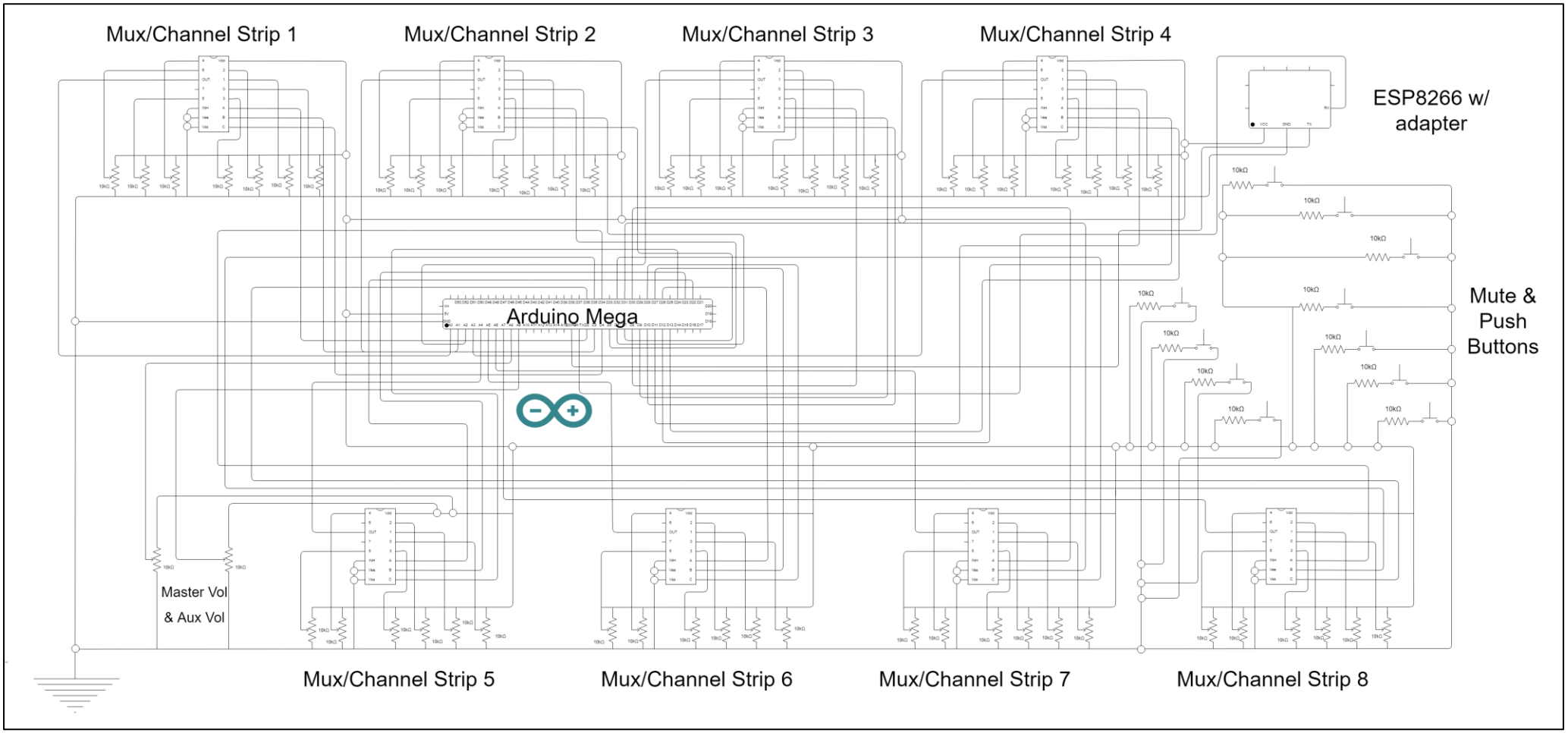


CSAW SketchUp – Side view

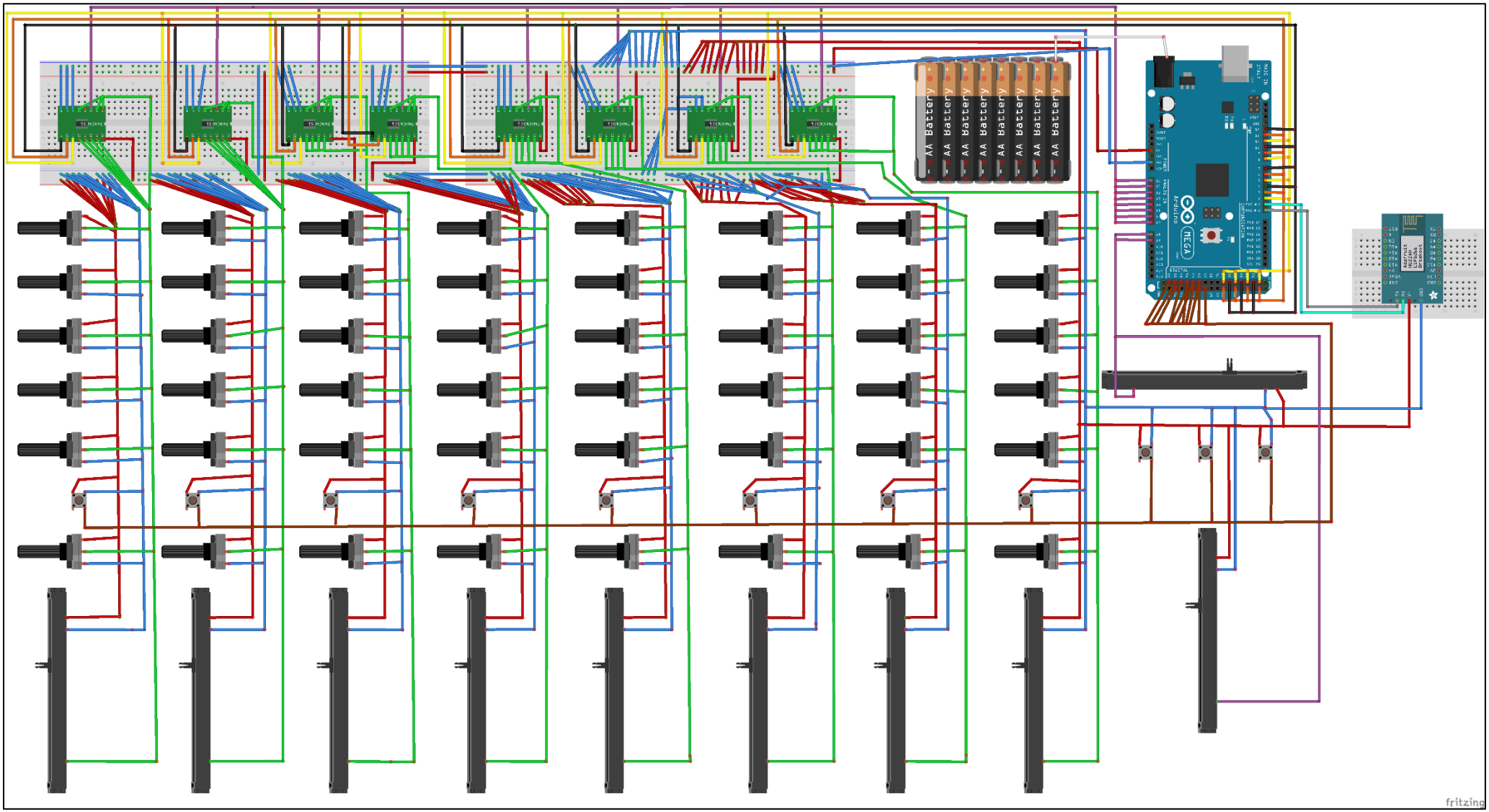
Appendix C.8 Final System/Measurements Diagram



Appendix C.9 Circuit Diagram



Appendix C.10 Fritzing Schematic



	VCC	5V voltage connections
	GND	Ground connections
	DATA	Arduino data inputs
	MUX - C	MUX analog inputs connected to sensors
	MUX - SIG / AN	MUX outputs / Arduino analog inputs
	MUX - S0	MUX Control Pin 0/A
	MUX - S1	MUX Control Pin 1/B
	MUX - S2	MUX Control Pin 2/C
	TX/RX	Arduino TX serial to ESP8266 RX connection
	RX/TX	Arduino RX serial to ESP8266 TX connection
	POWER	Arduino battery power

Appendix C.11 Pin Connections Table

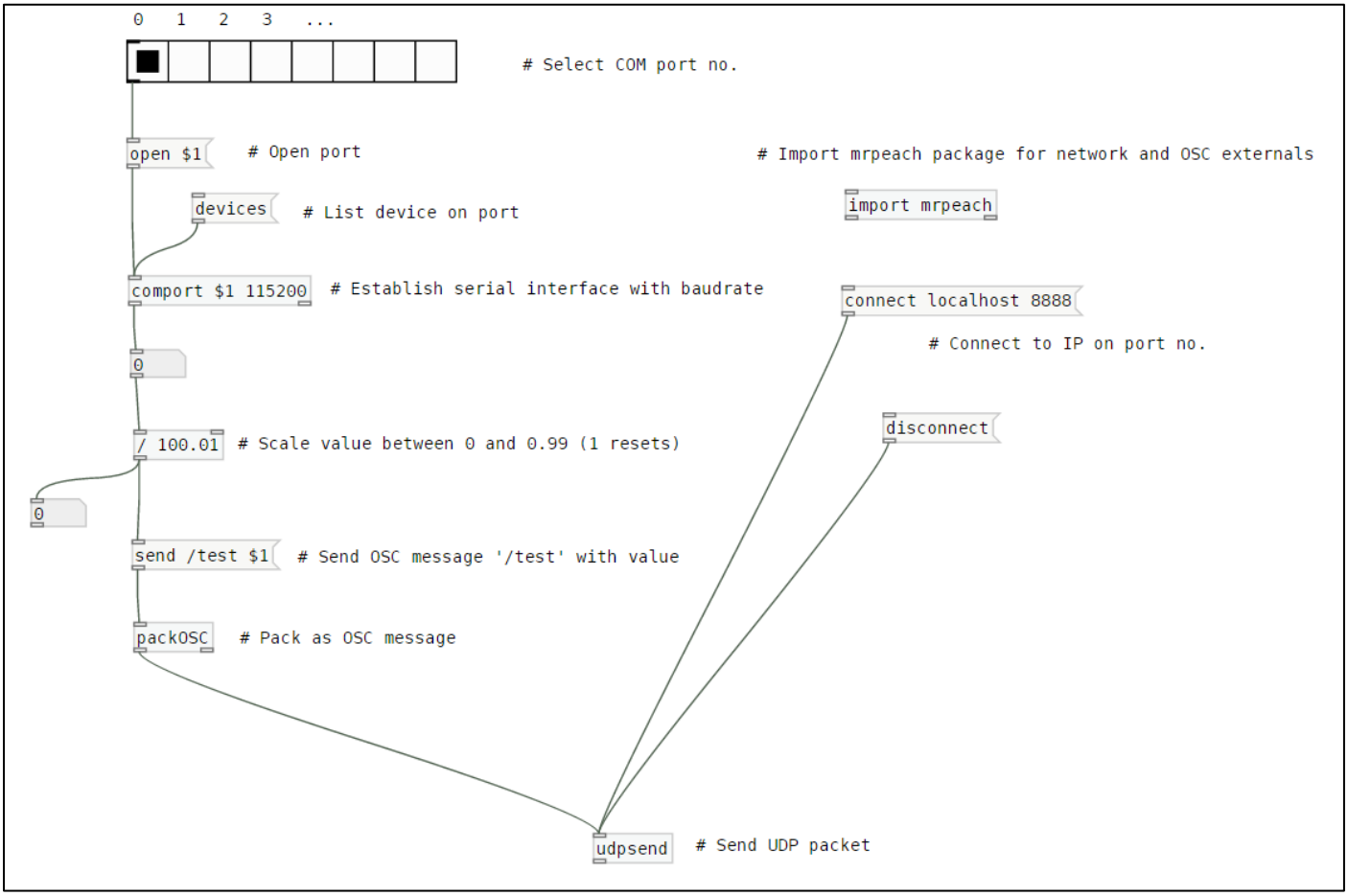
Component	Pin	Connection
Arduino Mega	5V	Breadboard positive rail
	GND	Breadboard ground rail
	A0 through A7	COM OUT pins of multiplexers
	D0 / RX	ESP8266 TX pin
	D1 / TX	ESP8266 RX pin
	D2, D3, D4	Mux 1 select pins (A, B, C)
	D5, D6, D7	Mux 2 select pins
	D8, D9, D10	Mux 3 select pins
	D11, D12, D13	Mux 4 select pins
	D22, D23, D24	Mux 5 select pins
	D26, D27, D28	Mux 6 select pins
	D30, D31, D32	Mux 7 select pins
	D34, D35, D36	Mux 8 select pins
ESP8266	VCC	Breadboard positive rail
	GND	Breadboard ground rail
	TX	Arduino Mega D0 / RX pin
	RX	Arduino Mega D1 / TX pin
Multiplexer (all)	Vdd	Breadboard positive rail
	INH	Breadboard ground rail
	Vee	Breadboard ground rail
	Vss	Breadboard ground rail
	COM OUT	A0 through A7 of Arduino
	A, B, C	Arduino data pins (see above)
	0 through 6 (5 after first 4 channels)	Center pins of rotary pots or DTB pin of slide pots
Potentiometer (slide)	DTB	0 through 6 (5) pins of mux
	VCC	Breadboard positive rail
	GND	Breadboard negative rail
Potentiometer (rotary)	Left pin	Breadboard positive rail
	Center pin	0 through 6 (5) pins of mux
	Right pin	Breadboard negative rail
Push buttons	Top-left	N/A
	Top-right	D41 though D52 of Arduino
	Bottom-left	Breadboard positive rail
	Bottom-right	Breadboard negative rail (via 10kΩ resistor)

Appendix D: Implementation Sketches & Patches

Appendix D.1: "Vol_Control"

Vol_Control

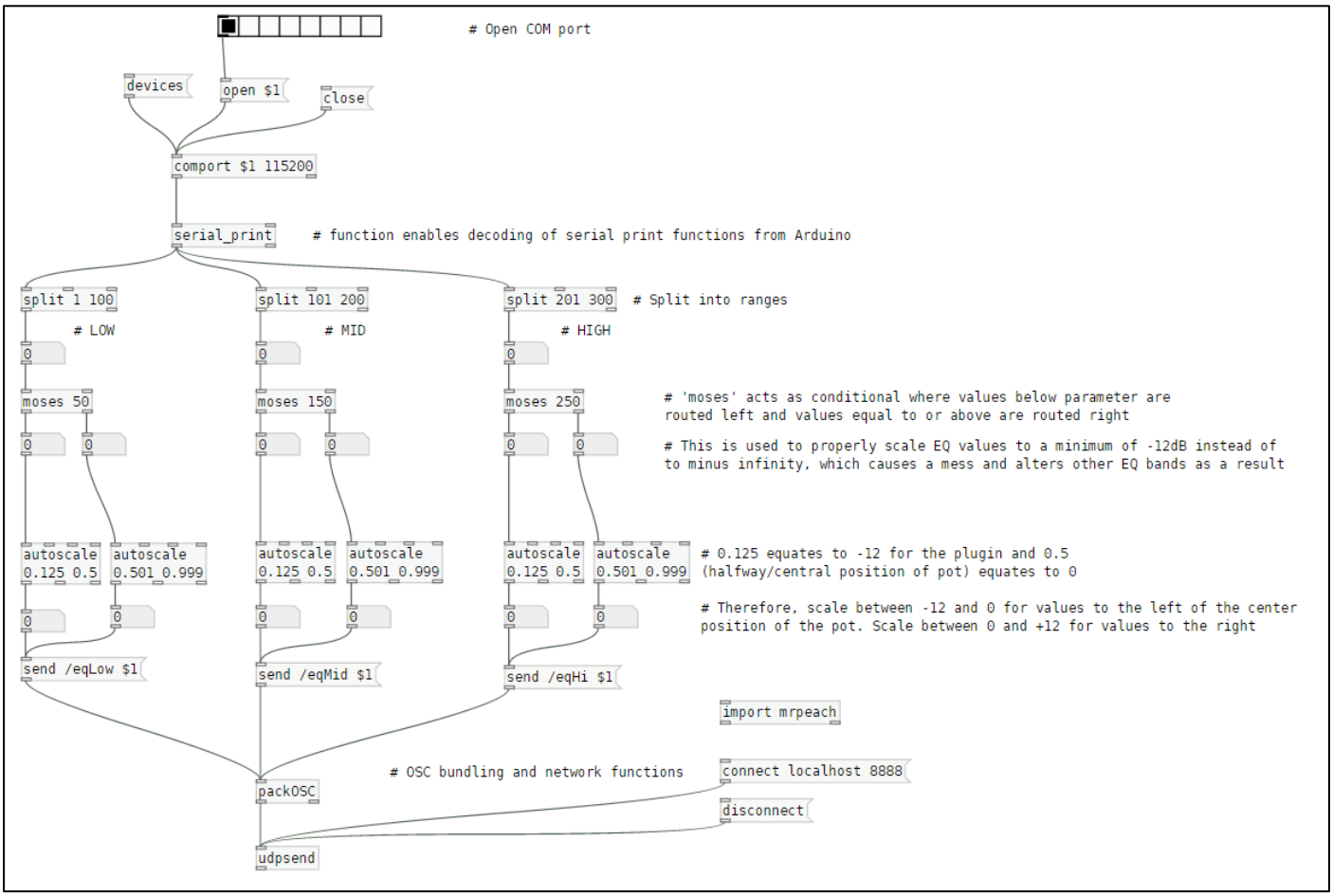
```
/***Vol_Control***/
// A simple volume control sketch - scales and writes pot data to Serial bus

// Specify analog input pin and initialise its value
int sensorPin = A0;
int sensorValue = 0;

void setup() {
  Serial.begin(115200); // Baudrate determines data rate in bits per second for serial transmission
}

void loop() {
  sensorValue = analogRead(sensorPin); // read pin value
  sensorValue = map(sensorValue, 0, 1023, 0, 100); // scale it between 0 and 100
  Serial.write(sensorValue); // write the value to serial
}
```



Appendix D.2: “ThreeBand_EQ”

ThreeBand_EQ

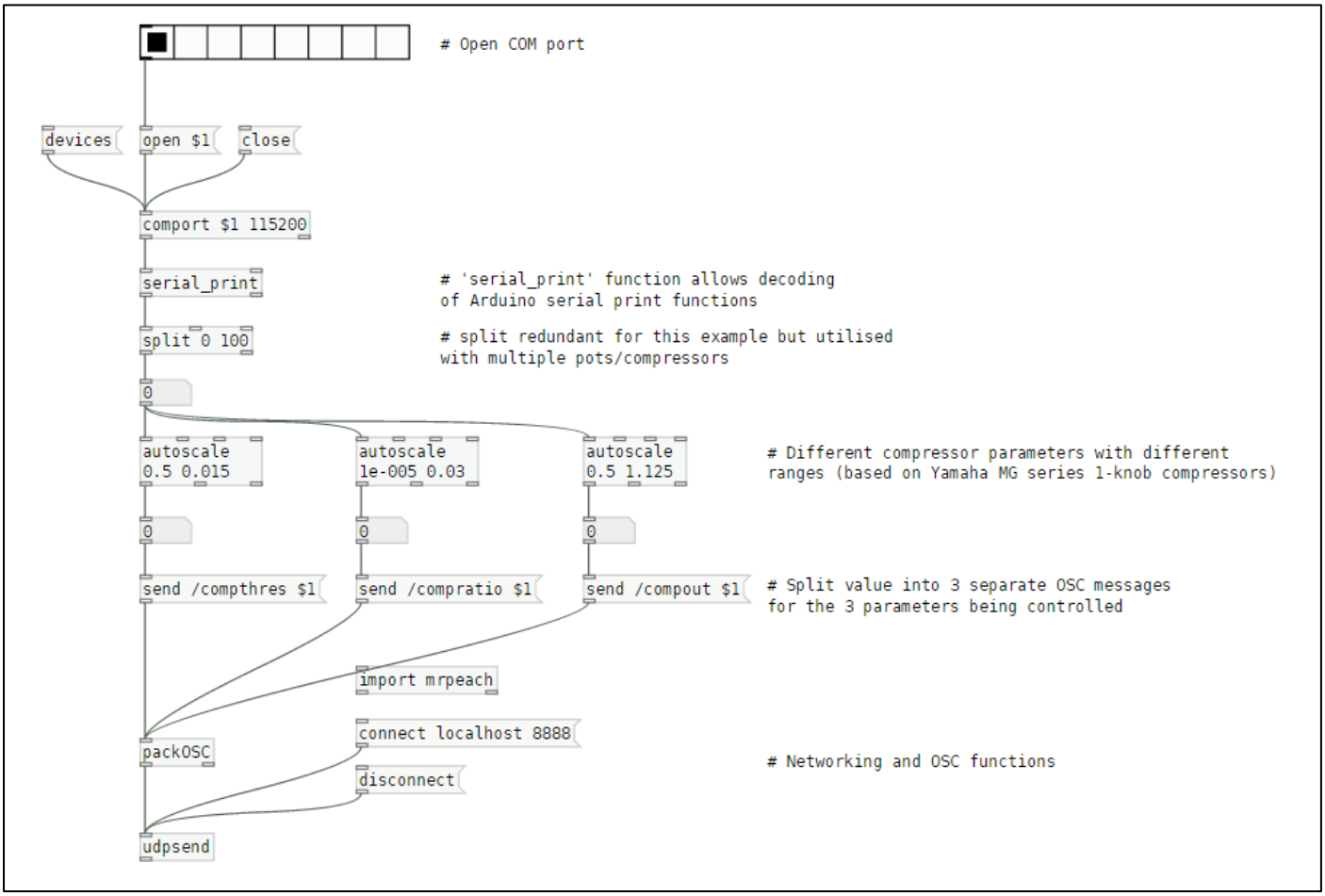
```
/***ThreeBand_EQ***/
// Three-band EQ control using 3 potentiometers : 3 analog pins

// Define & initialise analog pins
int pot1 = A0, pot2 = A1, pot3 = A3;
int pot1val = 0, pot2val = 0, pot3val = 0;

void setup() {
  Serial.begin(115200); // baud rate
}

void loop() {
  // Read values for each pot and scale between unique number range
  pot1val = analogRead(pot1);
  pot1val = map(pot1val,0,1023,1,100);

  pot2val = analogRead(pot2);
  pot2val = map(pot2val,0,1023,101,200);

  pot3val = analogRead(pot3);
  pot3val = map(pot3val,0,1023,201,300);

  // Print all values to serial where Pd can understand via 'serial_print'
  Serial.println(pot1val);
  Serial.println(pot2val);
  Serial.println(pot3val);
}
```



Appendix D.3: “OnePot_Compressor”

OnePot_Compressor

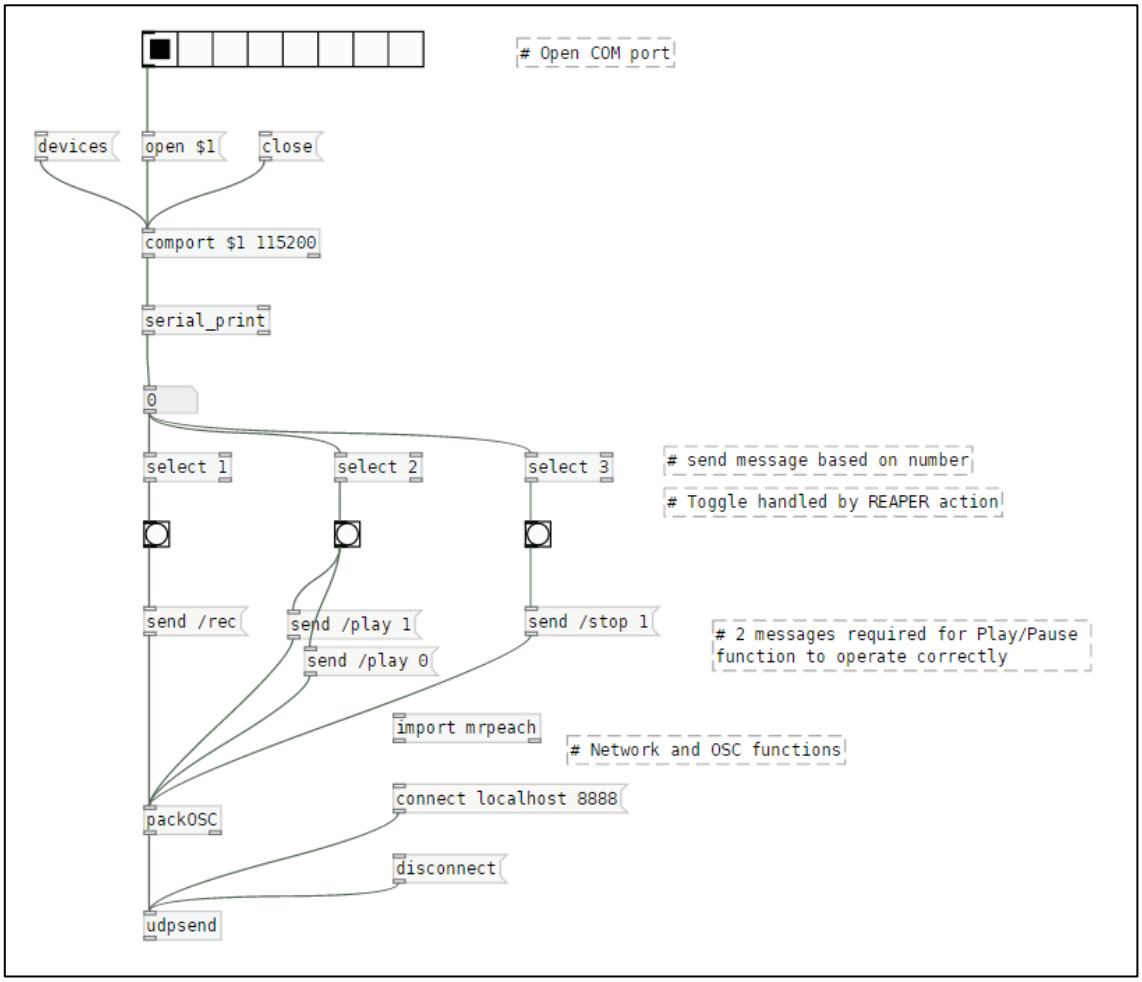
```
/***OnePot_Compressor***/
// One knob compressor based on Yamaha MG series
// One potentiometer controls compressor ratio, threshold and output gain

// Define and initialise pin and value
int sensorPin = A0;
int sensorValue = 0;

void setup() {
  Serial.begin(115200); // baud rate
}

void loop() {
  // Read value, scale between 0 and 100 and send to serial
  sensorValue = analogRead(sensorPin);
  sensorValue = map(sensorValue, 0, 1023, 0, 100);
  Serial.println(sensorValue);
}
```



Appendix D.4: “Transport_switches”

Transport_switches

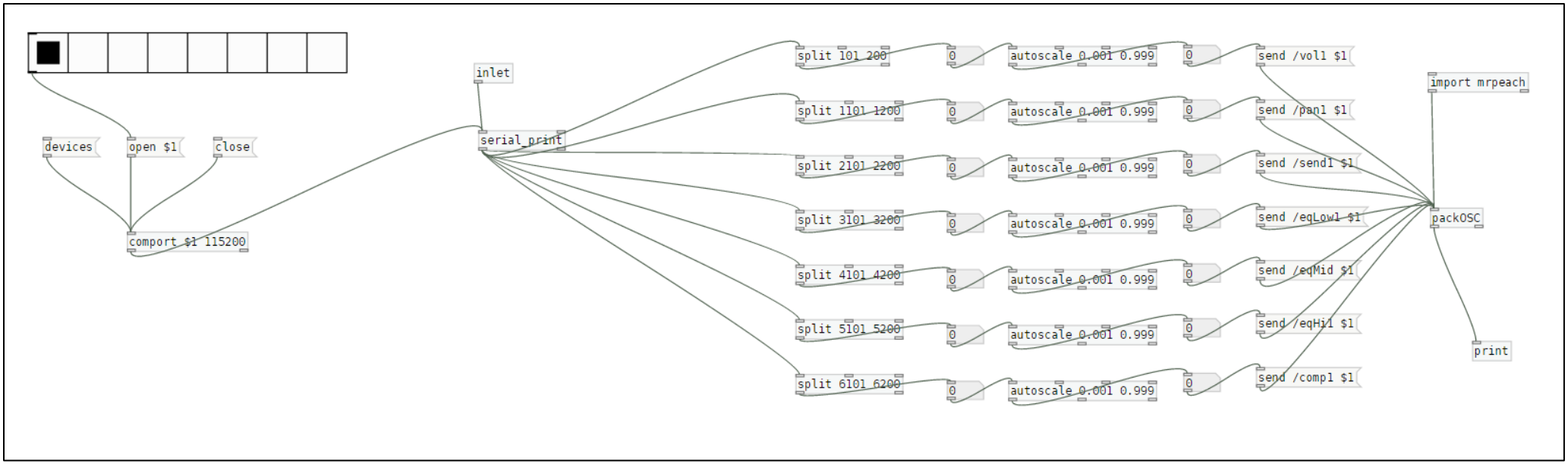
```
/***Transport_switches***/
// 3 button input switches that print specific numbers to serial on press for transport functionality

// Initialise pins used, state variable, previous reading variable and current reading
int chan1PIN = 2, state1 = LOW, previous1 = HIGH, reading1;
int chan2PIN = 3, state2 = LOW, previous2 = HIGH, reading2;
int chan3PIN = 4, state3 = LOW, previous3 = HIGH, reading3;

// Time variables for debouncing (ensures only a single signal is sent)
long time = 0, debounce = 200;

void setup()
{
  Serial.begin(115200);
  pinMode(chan1PIN, INPUT);
  pinMode(chan2PIN, INPUT);
  pinMode(chan3PIN, INPUT);
}

void loop()
{
  // read pin, toggle its state and print number to serial line
  // set time to time Arduino is active and change previous reading variable to current state
  reading1 = digitalRead(chan1PIN);
  if (reading1 == LOW && previous1 == HIGH && millis() - time > debounce) { Serial.println(0.91); time = millis(); }
  previous1 = reading1;

  reading2 = digitalRead(chan2PIN);
  if (reading2 == LOW && previous2 == HIGH && millis() - time > debounce) { Serial.println(0.92); time = millis(); }
  previous2 = reading2;

  reading3 = digitalRead(chan3PIN);
  if (reading3 == LOW && previous3 == HIGH && millis() - time > debounce) { Serial.println(0.93); time = millis(); }
  previous3 = reading3;
}
```



Appendix D.5: “Channel_Strip”

ChannelStrip §

```
/***ChannelStrip***/
// Channel Strip sketch for 7 potentiometers, using a CD4051BE mux
// -> scales all values and prints to Serial bus using println()

const int selectPins[3] = {2, 3, 4}; // S0~2, S1~3, S2~4
const int zInput = A0; // Connect common (Z) to A0 (analog input)

int start = 101;

void setup()
{
  Serial.begin(115200); // Initialize the serial port
  // Set up the select pins as outputs:
  for (int i=0; i<3; i++)
  {
    pinMode(selectPins[i], OUTPUT);
    digitalWrite(selectPins[i], HIGH);
  }
  pinMode(zInput, INPUT); // Set up Z as an input
}

void loop()
{
  // Loop through first seven of the eight pins.
  for (byte pin=0; pin<=6; pin++)
  {
    selectMuxPin(pin); // Select one at a time
    int inputValue = analogRead(A0); // and read Z
    inputValue = map(inputValue, 0, 1023, start, start+99);
    Serial.println(inputValue);
    start = start+1000;
  }
  Serial.println();
  delay(100);
  start = 101;
}

// The selectMuxPin function sets the S0, S1, and S2 pins
// accordingly, given a pin from 0-7.
void selectMuxPin(byte pin)
{
  for (int i=0; i<3; i++)
  {
    if (pin & (1<<i))
      digitalWrite(selectPins[i], HIGH);
    else
      digitalWrite(selectPins[i], LOW);
  }
}
```


devices
open $1
close
comport $1 115200
inlet
serial_print
split 101 200
split 1101 1200
split 2101 2200
split 3101 3200
split 4101 4200
split 5101 5200
split 6101 6200
0
autoscale 0.001 0.999
send /vol1 $1
send /pan1 $1
send /send1 $1
send /eqLow1 $1
send /eqMid $1
send /eqHi1 $1
send /comp1 $1
import mrpeach
packOSC
print

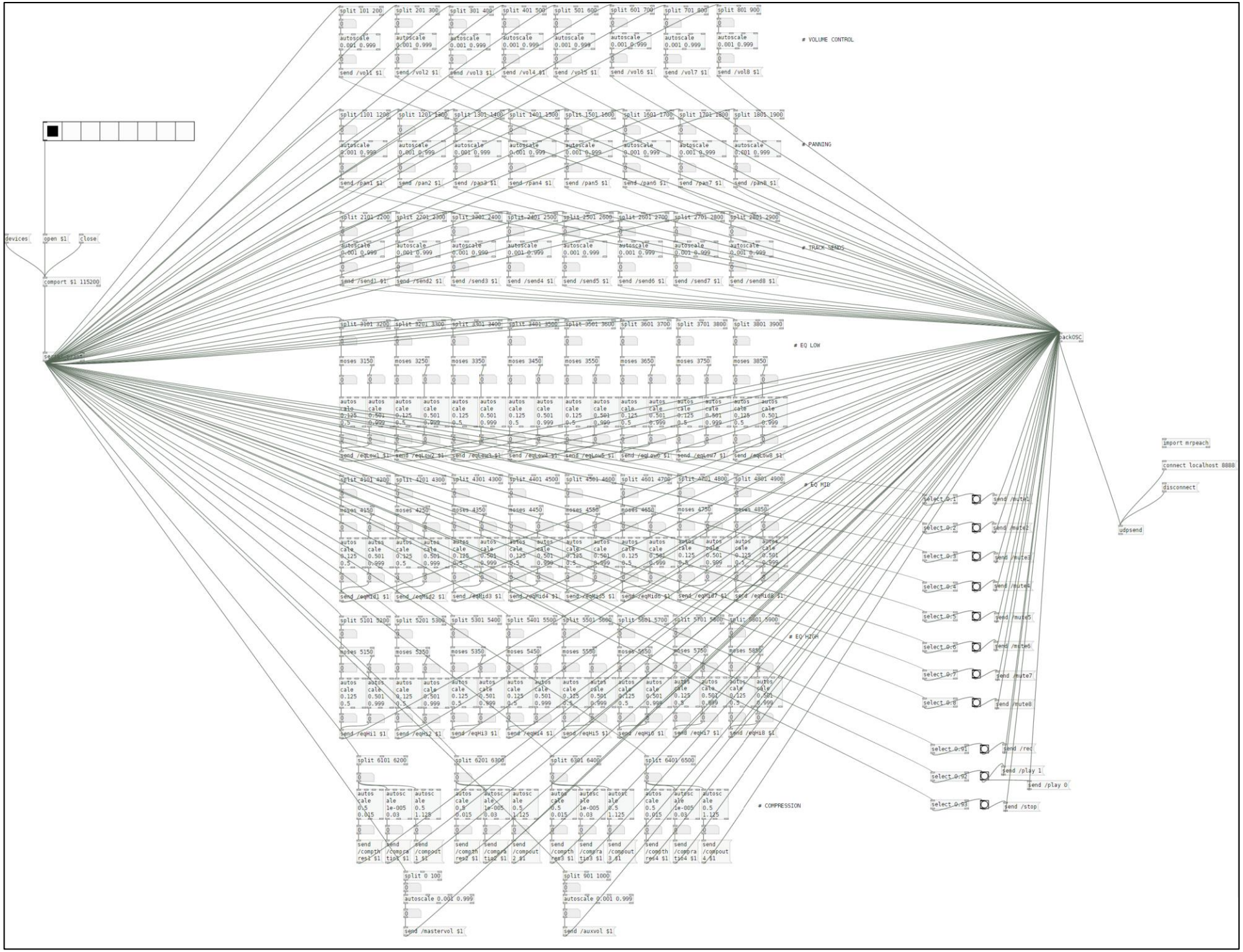
Appendix D.6: Final A Prototype

ArduinoMain

```
/***ArduinoMain***/
// Main Arduino Prototype A sketch for CSAW controller, using CD4051BE muxes
// Scales all pot values and responds to push buttons, printing to Serial bus using println()

// Initialise all mux pins
const int mux1Pins[3] = {2, 3, 4}; // S0=2, S1=3, S2=4
const int mux2Pins[3] = {5, 6, 7};
const int mux3Pins[3] = {8, 9, 10};
const int mux4Pins[3] = {11, 12, 13};
const int mux5Pins[3] = {22, 23, 24};
const int mux6Pins[3] = {26, 27, 28};
const int mux7Pins[3] = {30, 31, 32};
const int mux8Pins[3] = {34, 35, 36};

// Mux connections to Arduino analog inputs
const int mux1Input = A1; // Connect common OUT of Mux 1 to A1
const int mux2Input = A2;
const int mux3Input = A3;
const int mux4Input = A4;
const int mux5Input = A5;
const int mux6Input = A6;
const int mux7Input = A7;
const int mux8Input = A8;

// Button pin connections and state variables/previous reading/current reading
int mute1PIN = 41, state1 = LOW, previous1 = HIGH, reading1;
int mute2PIN = 42, state2 = LOW, previous2 = HIGH, reading2;
int mute3PIN = 43, state3 = LOW, previous3 = HIGH, reading3;
int mute4PIN = 44, state4 = LOW, previous4 = HIGH, reading4;
int mute5PIN = 45, state5 = LOW, previous5 = HIGH, reading5;
int mute6PIN = 46, state6 = LOW, previous6 = HIGH, reading6;
int mute7PIN = 47, state7 = LOW, previous7 = HIGH, reading7;
int mute8PIN = 48, state8 = LOW, previous8 = HIGH, reading8;
int recPIN = 50, stateR = LOW, previousR = HIGH, readingR;
int playPIN = 51, stateP = LOW, previousP = HIGH, readingP;
int stopPIN = 52, stateS = LOW, previousS = HIGH, readingS;

// Starting variable for pot ranges and debouncing variables for push buttons
int start = 101;
long time = 0, debounce = 200;

void setup()
{
  // Initialize the serial port with baud rate of 115200 bits per second
  Serial.begin(115200);

  // Set up the select pins as outputs:
  for (int i=0; i<3; i++)
  {
    pinMode(mux1Pins[i], OUTPUT);
    pinMode(mux2Pins[i], OUTPUT);
    pinMode(mux3Pins[i], OUTPUT);
    pinMode(mux4Pins[i], OUTPUT);
    pinMode(mux5Pins[i], OUTPUT);
    pinMode(mux6Pins[i], OUTPUT);
    pinMode(mux7Pins[i], OUTPUT);
    pinMode(mux8Pins[i], OUTPUT);
    digitalWrite(mux1Pins[i], HIGH);
    digitalWrite(mux2Pins[i], HIGH);
    digitalWrite(mux3Pins[i], HIGH);
    digitalWrite(mux4Pins[i], HIGH);
    digitalWrite(mux5Pins[i], HIGH);
    digitalWrite(mux6Pins[i], HIGH);
    digitalWrite(mux7Pins[i], HIGH);
    digitalWrite(mux8Pins[i], HIGH);
  }
```

```
  // Set up all pins to read inputs
  pinMode(mux1Input, INPUT);
  pinMode(mux2Input, INPUT);
  pinMode(mux3Input, INPUT);
  pinMode(mux4Input, INPUT);
  pinMode(mux5Input, INPUT);
  pinMode(mux6Input, INPUT);
  pinMode(mux7Input, INPUT);
  pinMode(mux8Input, INPUT);

  pinMode(mute1PIN, INPUT);
  pinMode(mute2PIN, INPUT);
  pinMode(mute3PIN, INPUT);
  pinMode(mute4PIN, INPUT);
  pinMode(mute5PIN, INPUT);
  pinMode(mute6PIN, INPUT);
  pinMode(mute7PIN, INPUT);
  pinMode(mute8PIN, INPUT);
  pinMode(recPIN, INPUT);
  pinMode(playPIN, INPUT);
  pinMode(stopPIN, INPUT);
}

void loop()
{
  //POTENTIOMETERS

  // Loop through first seven of the eight mux pins
  for (byte pin=0; pin<=6; pin++)
  {
    selectMuxPin(mux1Pins, pin); // Select one at a time
    int inputValue = analogRead(mux1Input); // Read OUT
    inputValue = map(inputValue, 0, 1023, start, start+99); // Scale it to range
    Serial.println(inputValue); // Print to serial

    selectMuxPin(mux2Pins, pin);
    inputValue = analogRead(mux2Input);
    inputValue = map(inputValue, 0, 1023, start+100, start+199);
    Serial.println(inputValue);

    selectMuxPin(mux3Pins, pin);
    inputValue = analogRead(mux3Input);
    inputValue = map(inputValue, 0, 1023, start+200, start+299);
    Serial.println(inputValue);

    selectMuxPin(mux4Pins, pin);
    inputValue = analogRead(mux4Input);
    inputValue = map(inputValue, 0, 1023, start+300, start+399);
    Serial.println(inputValue);

    selectMuxPin(mux5Pins, pin);
    inputValue = analogRead(mux5Input);
    inputValue = map(inputValue, 0, 1023, start+400, start+499);
    Serial.println(inputValue);

    selectMuxPin(mux6Pins, pin);
    inputValue = analogRead(mux6Input);
    inputValue = map(inputValue, 0, 1023, start+500, start+599);
    Serial.println(inputValue);

    selectMuxPin(mux7Pins, pin);
    inputValue = analogRead(mux7Input);
    inputValue = map(inputValue, 0, 1023, start+600, start+699);
    Serial.println(inputValue);

    selectMuxPin(mux8Pins, pin);
    inputValue = analogRead(mux8Input);
    inputValue = map(inputValue, 0, 1023, start+700, start+799);
    Serial.println(inputValue);

    start = start+1000;
  }

  // PUSH BUTTONS

  // Read pin, change state based on previous value and debounce
  // Set new time variable and set new previous value to current pin reading
  reading1 = digitalRead(mute1PIN);
  if (reading1 == LOW && previous1 == HIGH && millis() - time > debounce) { Serial.println(0.1); time = millis(); }
  previous1 = reading1;

  reading2 = digitalRead(mute2PIN);
  if (reading2 == LOW && previous2 == HIGH && millis() - time > debounce) { Serial.println(0.2); time = millis(); }
  previous2 = reading2;

  reading3 = digitalRead(mute3PIN);
  if (reading3 == LOW && previous3 == HIGH && millis() - time > debounce) { Serial.println(0.3); time = millis(); }
  previous3 = reading3;

  reading4 = digitalRead(mute4PIN);
  if (reading4 == LOW && previous4 == HIGH && millis() - time > debounce) { Serial.println(0.4); time = millis(); }
  previous4 = reading4;
```

```
  reading5 = digitalRead(mute5PIN);
  if (reading5 == LOW && previous5 == HIGH && millis() - time > debounce) { Serial.println(0.5); time = millis(); }
  previous5 = reading5;

  reading6 = digitalRead(mute6PIN);
  if (reading6 == LOW && previous6 == HIGH && millis() - time > debounce) { Serial.println(0.6); time = millis(); }
  previous6 = reading6;

  reading7 = digitalRead(mute7PIN);
  if (reading7 == LOW && previous7 == HIGH && millis() - time > debounce) { Serial.println(0.7); time = millis(); }
  previous7 = reading7;

  reading8 = digitalRead(mute8PIN);
  if (reading8 == LOW && previous8 == HIGH && millis() - time > debounce) { Serial.println(0.8); time = millis(); }
  previous8 = reading8;

  readingR = digitalRead(recPIN);
  if (readingR == LOW && previousR == HIGH && millis() - time > debounce) { Serial.println(0.91); time = millis(); }
  previousR = readingR;

  readingP = digitalRead(playPIN);
  if (readingP == LOW && previousP == HIGH && millis() - time > debounce) { Serial.println(0.92); time = millis(); }
  previousP = readingP;

  readingS = digitalRead(stopPIN);
  if (readingS == LOW && previousS == HIGH && millis() - time > debounce) { Serial.println(0.93); time = millis(); }
  previousS = readingS;

  int masterVol = analogRead(A0); // Read OUT
  masterVol = map(inputValue, 0, 1023, 0, 100); // Scale it to range
  Serial.println(masterVol);

  int sendVol = analogRead(A9);
  sendVol = map(inputValue, 0, 1023, 901, 1000);
  Serial.println(sendVol);

  Serial.println();
  delay(2000);
  start = 101;
}

// The selectMuxPin function sets the S0, S1, and S2 pins
// accordingly, given a pin from 0-7.
void selectMuxPin(int muxPins[], byte pin)
{
  for (int i=0; i<3; i++)
  {
    if (pin & (1<<i))
      digitalWrite(muxPins[i], HIGH);
    else
      digitalWrite(muxPins[i], LOW);
  }
}
```


split 101 200
split 201 300
split 301 400
split 401 500
split 501 600
split 601 700
split 701 800
split 801 900
autoscale 0.001 0.999
send /vol1 $1
send /vol2 $1
send /vol3 $1
send /vol4 $1
send /vol5 $1
send /vol6 $1
send /vol7 $1
send /vol8 $1
VOLUME CONTROL
split 1101 1200
split 1201 1300
split 1301 1400
split 1401 1500
split 1501 1600
split 1601 1700
split 1701 1800
split 1801 1900
send /pan1 $1
send /pan2 $1
send /pan3 $1
send /pan4 $1
send /pan5 $1
send /pan6 $1
send /pan7 $1
send /pan8 $1
PANNING
split 2101 2200
split 2201 2300
split 2301 2400
split 2401 2500
split 2501 2600
split 2601 2700
split 2701 2800
split 2801 2900
send /send1 $1
send /send2 $1
send /send3 $1
send /send4 $1
send /send5 $1
send /send6 $1
send /send7 $1
send /send8 $1
TRACK SENDS
devices
open $1
close
comport $1 115200
split 3101 3200
split 3201 3300
split 3301 3400
split 3401 3500
split 3501 3600
split 3601 3700
split 3701 3800
split 3801 3900
moses 3150
moses 3250
moses 3350
moses 3450
moses 3550
moses 3650
moses 3750
moses 3850
send /eqLow1 $1
send /eqLow2 $1
send /eqLow3 $1
send /eqLow4 $1
send /eqLow5 $1
send /eqLow6 $1
send /eqLow7 $1
send /eqLow8 $1
EQ LOW
split 4101 4200
split 4201 4300
split 4301 4400
split 4401 4500
split 4501 4600
split 4601 4700
split 4701 4800
split 4801 4900
moses 4150
moses 4250
moses 4350
moses 4450
moses 4550
moses 4650
moses 4750
moses 4850
send /eqMid1 $1
send /eqMid2 $1
send /eqMid3 $1
send /eqMid4 $1
send /eqMid5 $1
send /eqMid6 $1
send /eqMid7 $1
send /eqMid8 $1
EQ MID
split 5101 5200
split 5201 5300
split 5301 5400
split 5401 5500
split 5501 5600
split 5601 5700
split 5701 5800
split 5801 5900
moses 5150
moses 5250
moses 5350
moses 5450
moses 5550
moses 5650
moses 5750
moses 5850
send /eqHi1 $1
send /eqHi2 $1
send /eqHi3 $1
send /eqHi4 $1
send /eqHi5 $1
send /eqHi6 $1
send /eqHi7 $1
send /eqHi8 $1
EQ HIGH
split 6101 6200
split 6201 6300
split 6301 6400
split 6401 6500
send /compth res1 $1
send /compra tio1 $1
send /compout 1 $1
send /compth res2 $1
send /compra tio2 $1
send /compout 2 $1
send /compth res3 $1
send /compra tio3 $1
send /compout 3 $1
send /compth res4 $1
send /compra tio4 $1
send /compout 4 $1
COMPRESSION
split 0 100
autoscale 0.001 0.999
send /mastervol $1
split 901 1000
autoscale 0.001 0.999
send /auxvol $1
packOSC
udpsend
import mrpeach
connect localhost 8888
disconnect
select 0 1
send /mute1
select 0 2
send /mute2
select 0 3
send /mute3
select 0 4
send /mute4
select 0 5
send /mute5
select 0 6
send /mute6
select 0 7
send /mute7
select 0 8
send /mute8
select 0 91
send /rec
select 0 92
send /play 1
send /play 0
select 0 93
send /stop

Appendix E: User Testing Questionnaire

CSAW User Testing Questionnaire

- I consent that I voluntarily agree to participate in this questionnaire.
- I agree that all data I provide as part of the questionnaire may be used as part of this research study, the nature of which has been fully explained to me in detail.
- I understand that I can withdraw from partaking in the study and may retract my questionnaire at any time after the original session (in which case all data will be deleted).
- I also understand that I may decline to answer any part of the questionnaire if I wish.
- I accept that there will be no direct benefits from my participating in this questionnaire.
- I understand that all data I provide will be treated confidentially and anonymously, and only be used for the purposes of the agreed research study.
- I recognise that I can reach out at any time if I wish to access the information I have provided, either for review or withdrawal.

Signed: _____ Date: _____

SECTION ONE: PRELIMINARY BACKGROUND QUESTIONNAIRE

1. Which term best describes how you fit into the field of audio production? Please select one.

- ☐ Professional/Occupation – I work with audio/audio production as a profession
- ☐ Student – I study audio production in school/university
- ☐ Amateur hobbyist – I don't study/work with audio production, but I do it as a hobby
- ☐ Interested party – I don't consider audio production a hobby, but I still have an interest
- ☐ None - I don't work with or in the field of audio production
- ☐ Other (please specify): _____

2. What DAWs do you primarily use for manipulating/editing audio? Please tick all that apply:

- ☐ **Pro Tools**
- ☐ **Ableton Live**
- ☐ **Studio One**
- ☐ **REAPER**
- ☐ **Audacity**
- ☐ **GarageBand**
- ☐ **Logic Pro X**
- ☐ **FL Studio**
- ☐ **I don't use a DAW**
- ☐ **Other (please specify): _____**

3. Do you use an audio control surface or DAW controller for audio production? If yes, which make/model controllers? <u>Note</u>: this includes mixers/any controller that interact with DAW tasks.

_____
_____
_____
_____
_____
_____

4. Do you think that control surfaces can provide any benefit to audio production? Please explain why/why not.

_____
_____
_____
_____
_____
_____
_____
_____

5. "In the domain of digital audio, a control surface is a human interface device (HID) which allows the user to control a digital audio workstation or other digital audio application." (Wikipedia)

[iCON Platform X control surface]

Based on this definition, what functionality do you think is important for an audio control surface to have? Please rate the following criteria where:

(5=critical importance, 4=very important, 3=somewhat important,2=not really important, 1=not important, D=don't know)

☐ Volume control	**D**	**1**	**2**	**3**	**4**	**5**
☐ Panning	**D**	**1**	**2**	**3**	**4**	**5**
☐ Solo/Mute functionality	**D**	**1**	**2**	**3**	**4**	**5**
☐ Auxiliary sends	**D**	**1**	**2**	**3**	**4**	**5**
☐ Equalization & Compressor control	**D**	**1**	**2**	**3**	**4**	**5**
☐ Track controls (Record, Play, Stop etc.)	**D**	**1**	**2**	**3**	**4**	**5**
☐ WiFi functionality	**D**	**1**	**2**	**3**	**4**	**5**
☐ Mobile power/Portability	**D**	**1**	**2**	**3**	**4**	**5**
☐ Motorized faders	**D**	**1**	**2**	**3**	**4**	**5**
☐ LCD displays	**D**	**1**	**2**	**3**	**4**	**5**

6. Are there any control surface capabilities missing from the above list that you consider important? Please specify:

_____
_____
_____
_____

SECTION TWO: USER EXPERIENCE QUESTIONNAIRE

From the **System Usability Scale (John Brooke, 1986)**, rate the following 10 statements based on the system you just tested. Please check one box per statement only.

1. **I think that I would like to use this system frequently.**

Strongly disagree				Strongly agree
1	2	3	4	5

2. **I found the system unnecessarily complex.**

Strongly disagree				Strongly agree
1	2	3	4	5

3. **I thought the system was easy to use.**

Strongly disagree				Strongly agree
1	2	3	4	5

4. **I think that I would need the support of a technical person to be able to use this system.**

Strongly disagree				Strongly agree
1	2	3	4	5

5. **I found the various functions in this system were well integrated.**

Strongly disagree				Strongly agree
1	2	3	4	5

6. **I thought there was too much inconsistency in this system.**

Strongly disagree				Strongly agree
1	2	3	4	5

7. **I would imagine that most people would learn to use this system very quickly.**

Strongly disagree				Strongly agree
1	2	3	4	5

8. **I found the system very cumbersome to use.**

Strongly disagree				Strongly agree
1	2	3	4	5

9. **I felt very confident using the system.**

Strongly disagree				Strongly agree
1	2	3	4	5

10. **I needed to learn a lot of things before I could get going with this system.**

Strongly disagree				Strongly agree
1	2	3	4	5

Comments (general feedback/ease of use/improvements/(un)necessary features etc.):

_____

_____

_____

_____

_____

_____

Bibliography

ADC Calculator. Retrieved from learningaboutelectronics.com on 13 June 2019: http://www.learningaboutelectronics.com/Articles/Analog-to-digital-conversion-ADC-calculator.php

AI-Thinker. (2015). *ESP-01 WiFi Module Version 1.0.*

Analog to Digital Conversion. Retrieved from sparkfun.com on 13 June 2019: https://learn.sparkfun.com/tutorials/analog-to-digital-conversion/all

Analog to Digital Converter. Retrieved from wikipedia.org on 13 June 2019: https://en.wikipedia.org/wiki/Analog-to-digital_converter

analogRead() reference. Retrieved from arduino.cc on 11 August 2019: https://www.arduino.cc/reference/en/language/functions/analog-io/analogread/

Arduino - Home. Retrieved from arduino.cc on 2 August 2019: https://www.arduino.cc/

Arduino Mega 2560 with ESP8266 (ESP-01) Wifi, AT Commands and Blynk. Retrieved from youtube.com on 26 July 2019: https://www.youtube.com/watch?v=YLKEZtLhfZo

Arduino MIDI Controllers. Retrieved from makeuseof.com on 26 July 2019: https://www.makeuseof.com/tag/arduino-midi-controllers/

Audio Control Surface. Retrieved from wikipedia.org on 10 June 2019: https://en.wikipedia.org/wiki/Audio_control_surface

Avid. (2014). *Pro Tools | S3 User Guide.*

Behringer. (2014). *X-TOUCH: Quick Start Guide.*

Behringer. (2019). *Xenyx X1222USB User Manual.*

Blynk. Retrieved from blynk.io on 15 July 2019: https://blynk.io/

Brooke, J. (1986). *SUS - A quick and dirty usability scale.*

DAW Remote HD. (2019). Retrieved from eumlab.com on 11 August 2019: http://eumlab.com/daw-remote-hd/

Differences between Black-box testing and White-box testing. Retrieved from softwaretestingfundamentals.com on 10 August 2019: http://softwaretestingfundamentals.com/differences-between-black-box-testing-and-white-box-testing/

digitalWrite() reference. Retrieved from arduino.cc on 11 August 2019: https://www.arduino.cc/reference/en/language/functions/digital-io/digitalwrite/

Draw.io - Flowchart Maker & Online Diagram Software on 3 July 2019: https://www.draw.io/

electronics-tutorials.ws. Retrieved from electronics-tutorials.ws on 7 August 2019: https://www.electronics-tutorials.ws/logic/pull-up-resistor.html

ESP8266 Overview. Retrieved from espressif.com on 10 August 2019: https://www.espressif.com/en/products/hardware/esp8266ex/overview

Espressif Systems. (2018). *ESP8266EX Datasheet.*

Firmata reference. Retrieved from arduino.cc on 27 July 2019: https://www.arduino.cc/en/reference/firmata

Freed, A., Schmeder, A., & Zbyszynski, M. (2007). *Open Sound Control - A flexible protocol for sensor networking.*

Fritzing. Retrieved from fritzing.org on 25 June 2019: https://fritzing.org

Functional and Non-Functional Requirements - Specifications and Types. Retrieved from altexsoft.com on 10 August 2019: https://www.altexsoft.com/blog/business/functional-and-non-functional-requirements-specification-and-types/

github.com/agraef. Retrieved from GitHub.com on 25 June 2019: https://github.com/agraef/purr-data/

github.com/alexdrymonitis/Arduino_Pd. Retrieved from GitHub.com on 11 August 2019: https://github.com/alexdrymonitis/Arduino_Pd

github.com/CNMAT/OSC. Retrieved from GitHub.com on 11 August 2019: https://github.com/CNMAT/OSC

github.com/nodemcu/nodemcu-flasher. Retrieved from GitHub.com on 27 July 2019: https://github.com/nodemcu/nodemcu-flasher

github.com/odonnellf/CSAW. Retrieved from GitHub.com on 11 August 2019: https://github.com/odonnellf/CSAW/

github.com/tttapa/Control-Surface. Retrieved from GitHub.com on 11 August 2019: https://github.com/tttapa/Control-Surface

Instruments, T. (1998). *CD405xB CMOS Single 8-Channel Analog Multiplexer/Demultiplexer datasheet.*

Introduction to OSC. Retrieved from opensoundcontrol.org on 10 August 2019: http://opensoundcontrol.org/introduction-osc

iZotope, & McLaughlin, S. (2014). *Mixing with iZotope: Principles, Tips and Techniques.*

Leider, C. N. (2004). *Digital Audio Workstation.*

Mackie Control Universal. Retrieved from Sound on Sound on 3 June 2019: https://www.soundonsound.com/reviews/mackie-control-universal

Make MIDI Controller Arduino. Retrieved from makeuseof.com on 26 July 2019: https://www.makeuseof.com/tag/make-midi-controller-arduino/

Multiplexer Breakout Hookup Guide. Retrieved from learn.sparkfun.com on 10 August 2019: https://learn.sparkfun.com/tutorials/multiplexer-breakout-hookup-guide/all

nexperia. (2015). *74HC4067; 74HCT4067 16-channel analog multiplexer/demultiplexer Product datasheet.*

Office Timeline. Retrieved from officetimeline.com on 5 June 2019: officetimeline.com

Ohm's Law. Retrieved from wikipedia.org on 10 August 2019: https://en.wikipedia.org/wiki/Ohm's_law

Opencircuit. (n.d.). *YwRobot Breadboard Power Supply MB-V2.*

OpenTransport. Retrieved from hacakday.io: https://hackaday.io/project/2983-opentransport

Potentiometer Taper. Retrieved from resistorguide.com on 11 August 2019: http://www.resistorguide.com/potentiometer-taper/

PreSonus. (2008). *FaderPort User's Manual Version 2.0.*

Puckette, M. S. (2016). *Pure Data.*

Pure Data. Retrieved from puredata.info on 11 August 2019: https://puredata.info/

REAPER - Digital Audio Workstation. Retrieved from reaper.fm on 10 August 2019: https://www.reaper.fm/

ReaPlugs VST FX Suite. Retrieved from reaper.fm on 11 August 2019: https://www.reaper.fm/reaplugs/

Royce, D. W. (1970). *Managing the Development of Large Software Systems.*

rtpMIDI. Retrieved from tobias-erichsen.de on 18 July 2019: http://www.tobias-erichsen.de/software/rtpmidi.html

Serial reference. Retrieved from arduino.cc on 30 July 2019: https://www.arduino.cc/reference/en/language/functions/communication/serial/

Serial.write() reference. Retrieved from arduino.cc on 30 July 2019: https://www.arduino.cc/en/Serial.Write

SignSpec | Sign & Display Solutions. Retrieved from signspec.ie on 12 August 2019: http://www.thesignspecialist.ie/

SketchUp: 3D Design Software | 3D Modeling on the Web. Retrieved from sketchup.com on 30 July 2019: https://www.sketchup.com/

SoftwareSerial reference. Retrieved from arduino.cc on 30 July 2019: https://www.arduino.cc/en/Reference/softwareSerial

Switch reference. Retrieved from arduino.cc on 10 August 2019: https://www.arduino.cc/en/tutorial/switch

TouchOSC. Retrieved from hexler.net on 18 July 2019: https://hexler.net/products/touchosc

Turchet, L., Fischione, C., Essl, G., Keller, D., & Barthet, M. (2018). Internet of Musical Things: Vision and Challenges. *IEEE Access*.

UML Activity Diagram - tutorialspoint.com. Retrieved from tutorialspoint.com on 4 July 2019: https://www.tutorialspoint.com/uml/uml_activity_diagram.htm

University of Alicante - Circuit Diagram. Retrieved from web.ua.es/docivis/magnet/circuit_diagram.html on 11 August 2019: https://web.ua.es/docivis/magnet/circuit_diagram.html

Välimäki, V., & Reiss, J. D. (2016). All About Audio Equalization: Solutions and Frontiers. *MDPI Applied Sciences*.

Wright, M., Freed, A., & CNMAT. (1997). *Open Sound Control: A New Protocol for Communicating with Sound Synthesizers.*

Yamaha. (2014). *MG20XU / MG20 / MG16XU / MG16 / MG12XU / MG12 Owner's Manual.*