

# Python HW Link

## High Level Overview

As Flight Simulators increasingly offer support for API interfaces (e.g. DCS, P3D), it now is feasible to move away from using Joystick interfaces (along with fun of dealing with USB issues) to a distributed system using Ethernet. Such an approach also simplifies output interfacing (e.g. indicators and gauges) as tasks and processes can be divided up across a number of devices and technologies.

This project builds on the Python/Raspberry Pi project used in the Huey Warning panel.

# Table of Contents

<u>High Level Overview</u> .....	1
<u>Definitions</u> .....	4
<u>Principles</u> .....	4
<u>IP Addressing</u> .....	6
<u>Hardware Approach</u> .....	6
<u>Printed Circuit Boards</u> .....	6
<u>Design and Manufacturing of the PCB</u> .....	7
<u>PCBs Used in the Project</u> .....	18
<u>Arduino Output</u> .....	19
<u>Output PCB</u> .....	24
<u>Panel Switch PCB</u> .....	26
<u>Small PCB</u> .....	28
<u>Arduino Digital Connector</u> .....	29
<u>Arduino_NW_Template</u> .....	31
<u>Testing the PCBs</u> .....	35
<u>Interfacing to the Sims</u> .....	37
<u>P3d</u> .....	37
<u>DCS</u> .....	40
<u>X-Plane</u> .....	47
<u>Choice of platform for the hardware interface</u> .....	50
<u>Program tasks</u> .....	51
<u>Protocols</u> .....	51
<u>Learning's from Different Modules</u> .....	52
<u>Determining which Simulator is Operational</u> .....	52
<u>Raspberry Pi</u> .....	53
<u>Maintaining the Pi</u> .....	53
<u>Initiating Scripts on Pi Nodes</u> .....	53
<u>Receive UDP Port utilisation</u> .....	54
<u>System Flows</u> .....	57
<u>Code Modules</u> .....	59
<u>General Sim 7219</u> .....	59
<u>Hornet Altimeter</u> .....	59
<u>PyHWLink Arduino Sender Emulator</u> .....	60
<u>PyHWLink Display Output Emulator</u> .....	61
<u>PyHWLink GUI Sender</u> .....	62
<u>PyHWLink Lamp Output Emulator</u> .....	64
<u>PyHWLink Keystroke Sender</u> .....	65
<u>PyHWLink Pri_Node_Input</u> .....	66
<u>PyHWLink Pri_Node_Output</u> .....	67
<u>pyHWLink Radio Control</u> .....	67
<u>pyHWLink Serial</u> .....	68

<u>pyHWLink USB Reader.....</u>	71
<u>USB Interfaces.....</u>	72
<u>SimConnect To IP.....</u>	74
<u>PyHWLink Control XPlane.....</u>	75
<u>UDP Input Control.....</u>	77
<u>UDP Reflector.....</u>	78
<u>Code Status.....</u>	79
<u>Coding Learnings.....</u>	79
<u>Logging.....</u>	79
<u>Command Line Parameters.....</u>	81
<u>Configuration Files.....</u>	83
<u>Error Handling In Python.....</u>	84
<u>Persisting Information.....</u>	85
<u>Networking in Python.....</u>	86
<u>Pick a Python Version.....</u>	88
<u>Processing Data.....</u>	89
<u>Working with Datasets.....</u>	90
<u>Working with Dictionaries or Dictionaries.....</u>	91
<u>Working with a GUI.....</u>	92
<u>PI Addressing.....</u>	94
<u>Interfacing to a GPS.....</u>	95
<u>The Development and Test Environments.....</u>	95
<u>The Sim PC – making it usable.....</u>	96
<u>Remote Control.....</u>	96
<u>The Firewall.....</u>	96
<u>Troubleshooting If Data isn't being received.....</u>	97
<u>Hardware Details.....</u>	97
<u>Arduino Ethernet Shields – Attack of the Clones.....</u>	97
<u>Building the Panels.....</u>	100
<u>Open Hornet.....</u>	101
<u>Fusion 360 – Bodies and Components.....</u>	101
<u>Working with STL Files.....</u>	101
<u>Working with DXF in Visio.....</u>	102
<u>Graphical OLED – SSD1306.....</u>	102
<u>Building the Panels.....</u>	105
<u>Panel Construction.....</u>	105
<u>Panel Design.....</u>	105
<u>Merging Diagrams into Large Panel Diagrams.....</u>	107
<u>Panel Pre Cutting Checklist.....</u>	108
<u>Panel Cutting.....</u>	108
<u>Documenting the Project.....</u>	110
<u>Sharing Progress Updates.....</u>	110
<u>System Build.....</u>	110
<u>Hardware To Do's.....</u>	111
<u>Code To Do's.....</u>	111

## Definitions

Primary Flight Simulator – the PC running an instance of the flight simulator  
Primary Node – the Raspberry Pi node that communicates to the Primary Flight Simulator  
Distributed Node – either a Raspberry Pi or Arduino (with Ethernet shield) that communicates to the Primary node.

## Principles

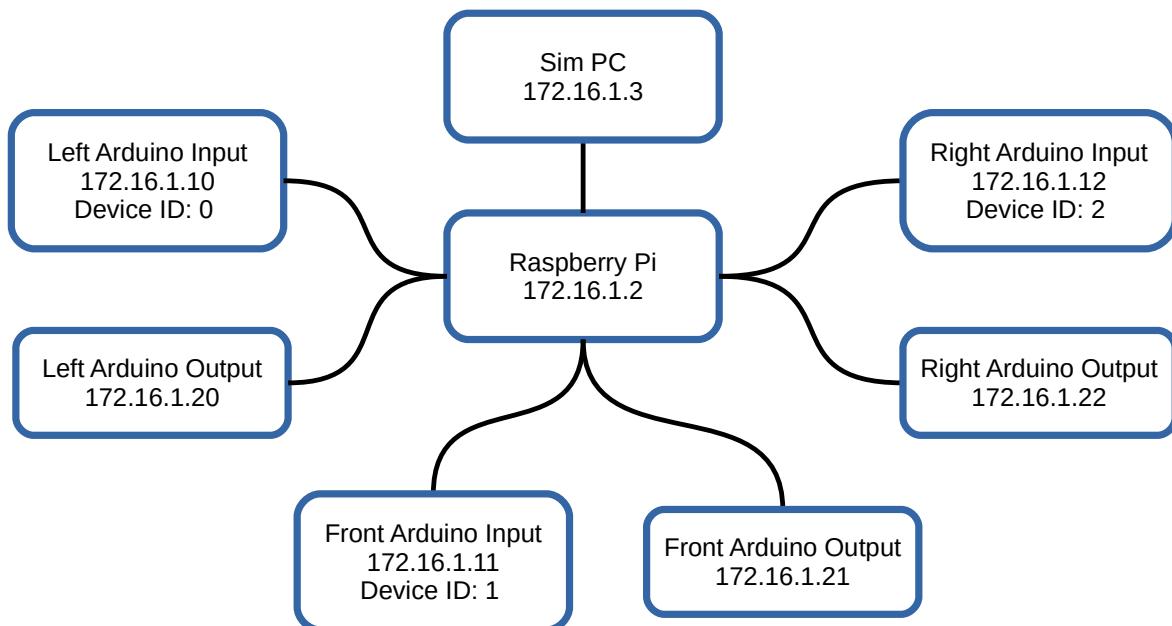
- Use UDP for all communications where possible. This removes any possible performance issues associated with Nagle and TCP slow start. It also means components are loosely coupled, enabling them to be restarted without impacting other modules.
- Accept inputs from push, toggle, rotary, and rotary encoders.
- Outputs – analog and digital, text. All outputs are normalised before being sent to output card/block. Should consider the format used by DCS-BIOS
- Multiple Sim support. As outputs are normalised the and loosely coupled now simulator support can be added without negatively impacting existing sim support. The receive interface from the Sim listens on unique ports, allowing the code to be running at all times.
- A shallow native shim is used to link the simulator to this hardware modules. As an example, LUA is used with DCS, for P3D Sim Connect used.
- Remote shutdown of all Pi nodes is provided through the master, once this has been invoked the nodes will shut down the OS, requiring either a hardware reset, and a power cycle to resume. Ideally outputs will display a checker board to reflect a shutdown command has been received.
- Indicator test, a single command will be supported to light all indicators, and perhaps cycle gauges.
- Downstream nodes should accept a request to report input switch position. On receipt of such a request the node will send a report of switch positions, probably at a rate of 20 per second. Need to consider reporting toggle switches in off position and three position switches.
- Will initially develop using DCS, and the variable names currently used by the A10
- The IP addressing of the ‘internal’ network (i.e. between the primary Flight Simulator computer) as well as between nodes will use the 172.16.1.X network. This enabled multiple flight simulators to share a common 192.168.X.X network. If there is only a single simulator the 172.16.1.X network can exist as a secondary address on the Primary network interface, if multiple Simulators share a network, then the internal network should use a different network interface on the Primary Computer.
- Mapping of physical inputs to simulator functions is performed on the Primary Pi node. This keeps the distributed nodes independent of flight sim, and relatively simple,

enabling the use Arduino nodes as needed without adding unneeded complexity.

- Each distributed node will have a Unique identifier, which largely is used to uniquely identify different input modules.
- Each distributed node will maintain a state machine for its interfaces, sending only deltas to the Primary Node
- Distributed nodes receiving non-string values receive data as A=V:A1=V1:A2=V2
- Packets from input nodes will have the format of DXX,XX:YYY:V,XX:YYY:V, where XX is the input node number, YYY is the digital input number, and V is the value. For analog inputs, the first character in YYY is an 'A'.
- If an AV pair is not known it will be silently discarded unless the Primary node is in learning mode – where it will ask the operator what task should be assigned to the unknown AV pair.
- Learning Mode is determined by an argument on the command line 'learning' or in the configuration file
- Debug Mode is determined by an argument on the command line 'debug'

## IP Addressing

### IP Addressing



Inputs – 172.16.1.10-17  
Outputs – 172.16.1.20-27

IP Addresses for Arduino devices are dynamically determined during startup by the three address select jumpers situated between ports D3 to D7 on the upper PCB.

## Hardware Approach

The simulator is divided into three zones, each linked using Ethernet and power. This enables the simulator to be easily assembled/disassembled without having to disconnect/reconnect a large number of cables.

As described later in this document the Primary interface to the Simulator is a Raspberry Pi which then interfaces inputs and outputs from Arduino with Ethernet Shields in the front, left, and right sections of the pit.

## Printed Circuit Boards

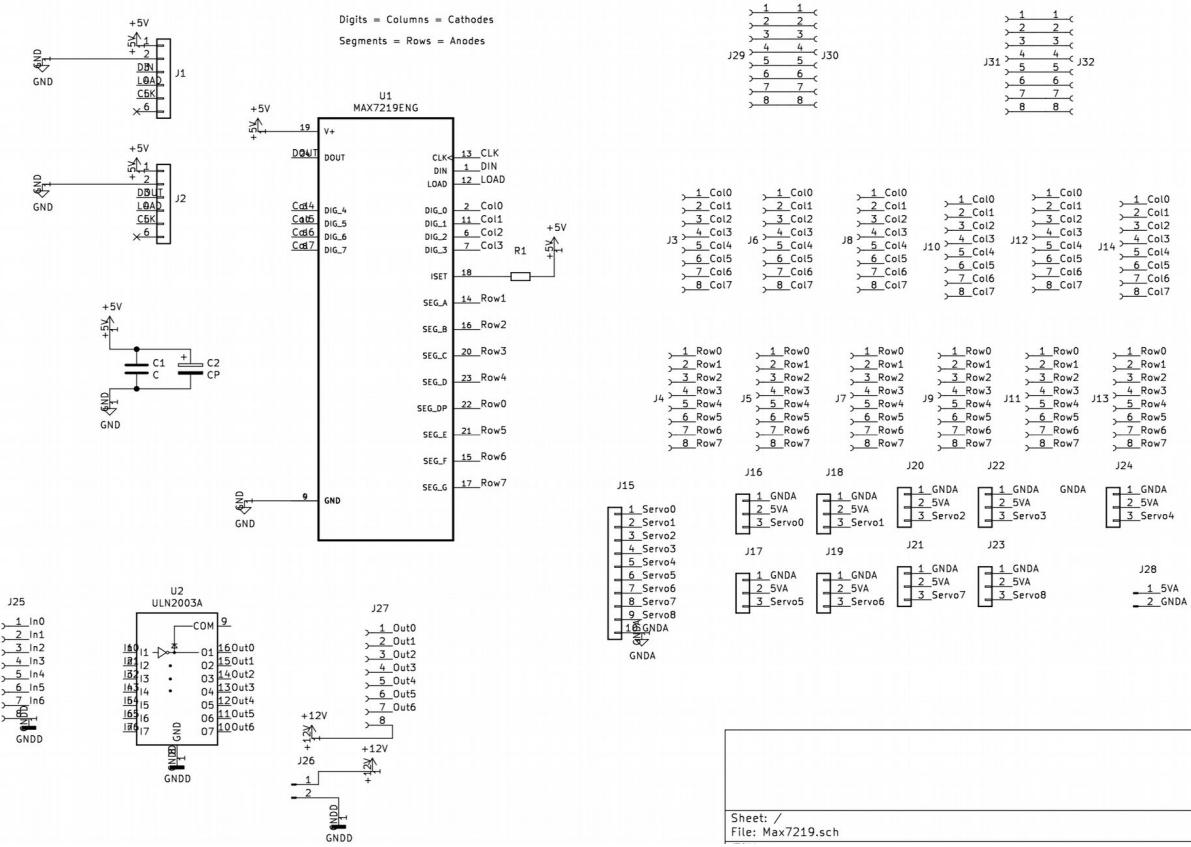
# Design and Manufacturing of the PCB

KiCad<sup>1</sup> was used for drawing schematics and PCB design. KiCad is an Open Source Electronics Design Automation Suite which enables you to quickly sketch up a circuit diagram, assign footprints to devices, and then layout a PCB. Whilst KiCad is Open Source – it is supported by donations managed by CERN<sup>2</sup> (the same group who fostered the development of the web and the large Hadron Collider). KiCad runs on both Mac and Windows.

One thing KiCad does not currently do is auto-routing of PCB tracks, which can get a little tedious after the first 100 tracks on a PCB. The good news there is a FreeRouting<sup>3</sup>, it is powered by Java (so you'll need to install Java Runtime).

## Using KiCad

Draw up the circuit diagram – remember to assign footprint to devices. Where there are a relatively number of connections to connections – its a good idea to use buses so the diagram is not overload with criss-crossing tracks.

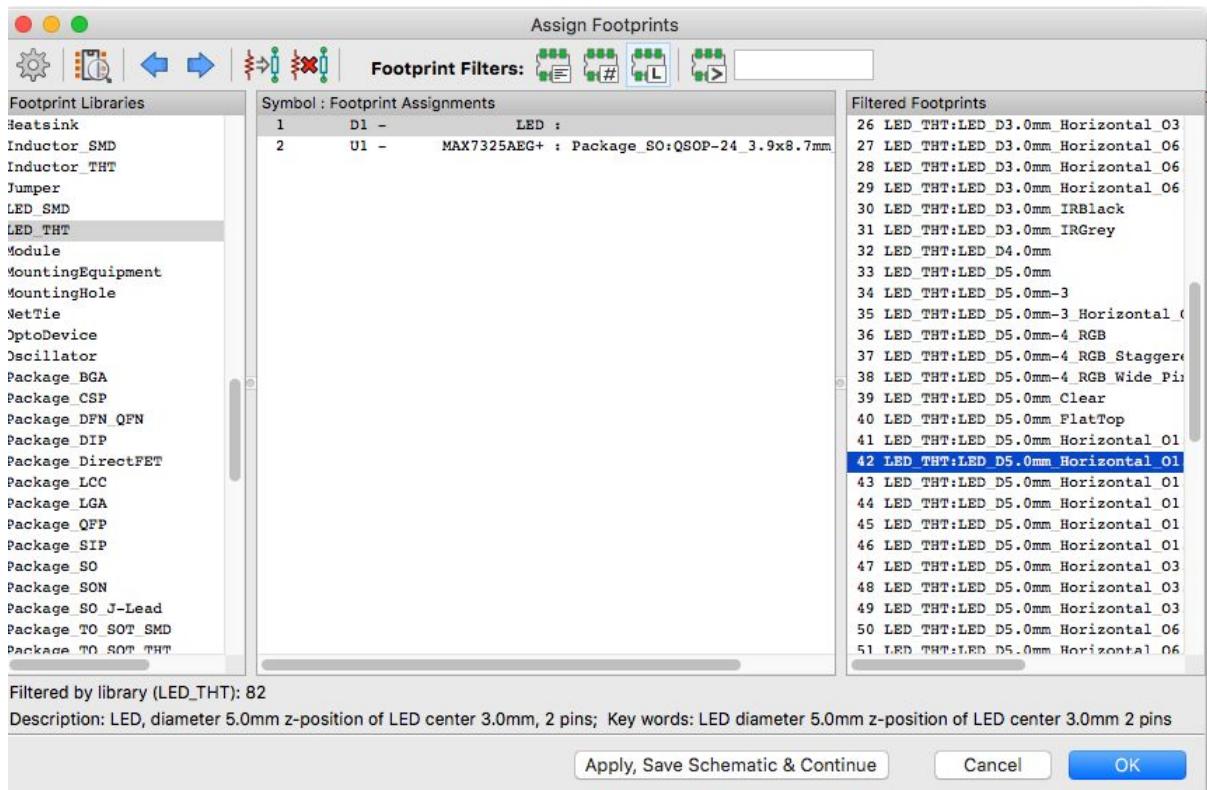


1 KiCad - <http://www.kicad-pcb.org/>

2 Donate to KiCad - <https://cernandsocietyfoundation.cern/projects/kicad-development>

3 Free Routing - <https://freerouting.org/freerouting/using-with-kicad>

Ensure all devices have a footprint assigned – Tools → Assign Footprints. Generally we'll be using THT (Through Hole) instead of SMD.



Once the drawing is complete, assign device numbers by annotating the schematic – Tools → Annotate Schematic. If there has been a bunch of adds/moves/etc it is not a bad idea to Reset Existing Annotations.

It is possible that the device you want to use is not included in the default KiCad library. I ran into this with the Max7219. If you are dealing with a new component, SnapEDA offers libraries that can be imported into KiCad. You do need to register and validate your email.

**Build Circuit Boards Faster with SnapEDA**

Download free symbols, footprints, and 3D models for millions of electronic components. Compatible with all major PCB tools.

Sign Up Free



**MAX7219**

Package Type: DIP-24  
CAD Models: [Symbol](#) and [Footprint](#)

[Add to Library](#) [See Datasheet PDF](#) [Shop Now](#)

**Symbol** 

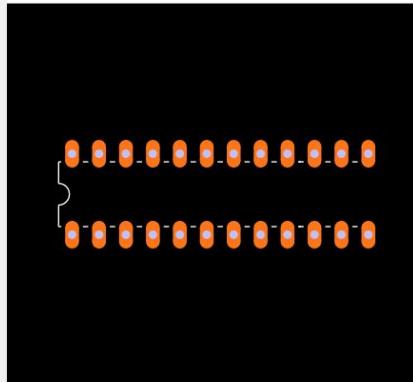
DIN	DOUT
DIG0	SEGDP
DIG1	SEGA
DIG2	SEGB
DIG3	SEGC
DIG4	SEGD
DIG5	SEGE
DIG6	SEGF
DIG7	SEGG
VCC	ISET
GND@1	LOAD
GND@2	CLK

MAX7219

MAX7219  Downloaded 240 times

[Download Symbol and Footprint](#)

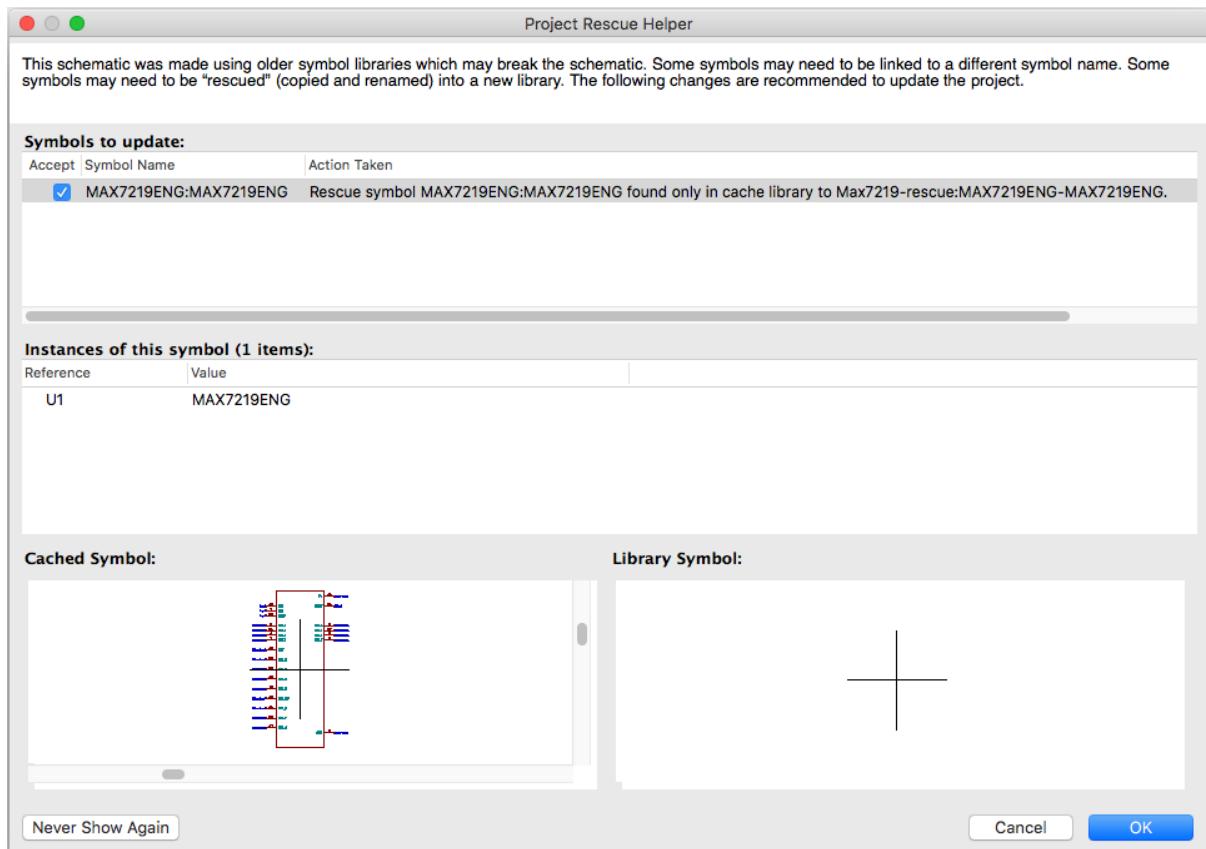
**Footprint** 



Default  Downloaded 264 times

[Download Footprint](#)

If you edit an existing design on a new computer without the component libraries installed you may get the following



The most simple thing to do if this happens is using the rescue symbol.

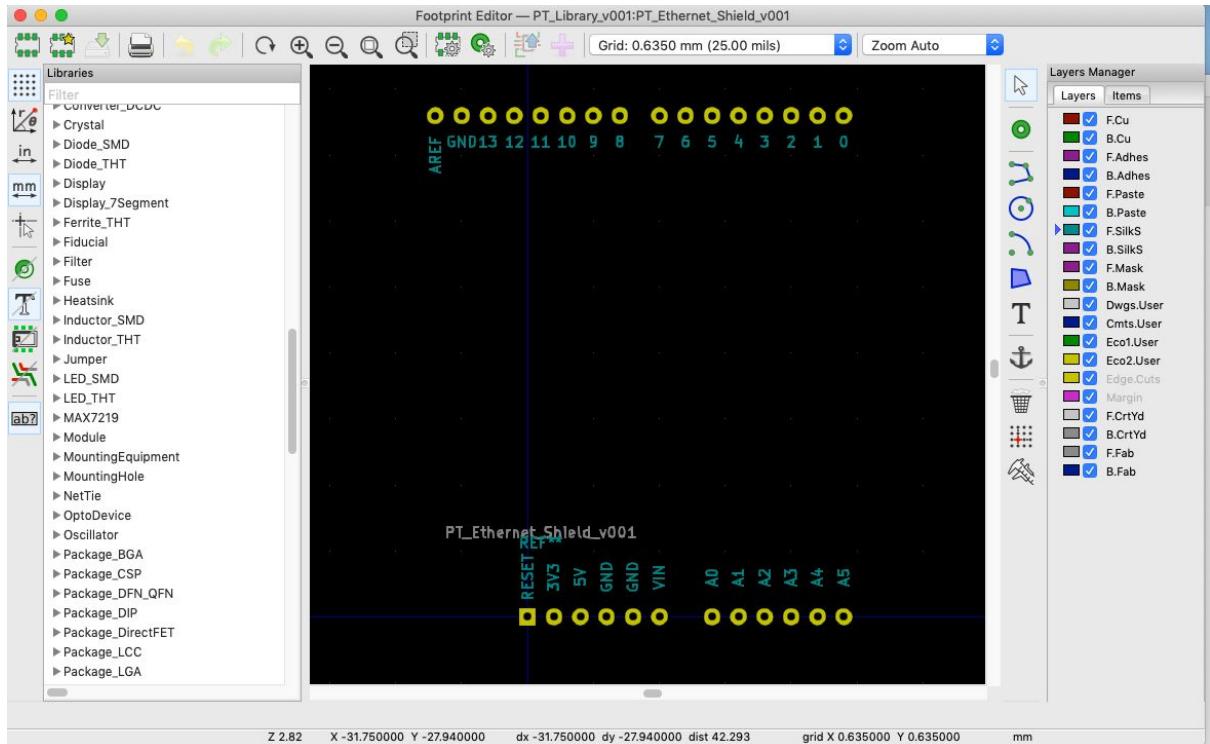
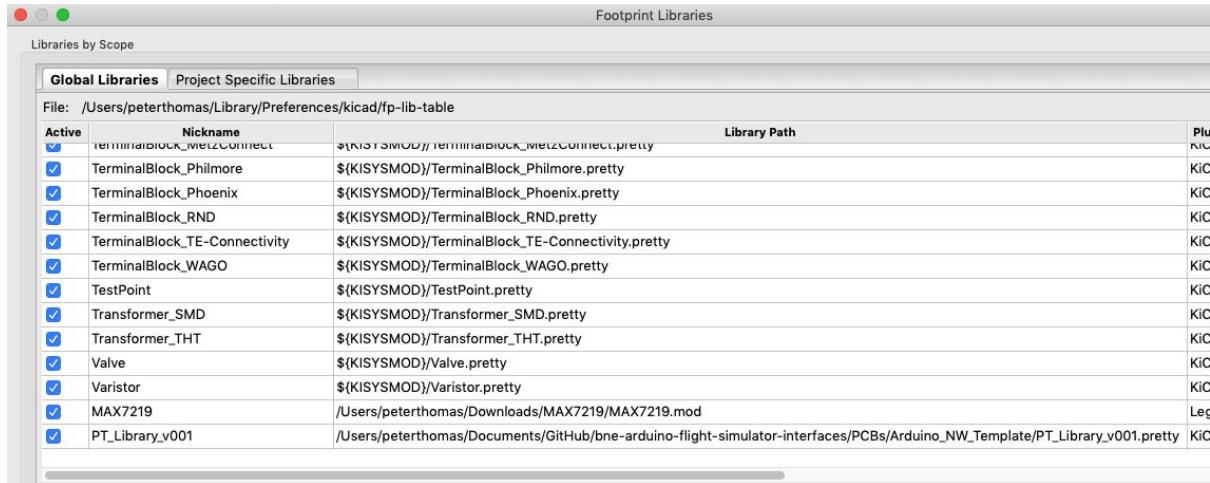
Once you've downloaded the zip file, add it to KiCad (SnapEDA will provide you with steps – which I've duplicated here)

In KiCad, go to *Tools > Edit PCB Footprints*.

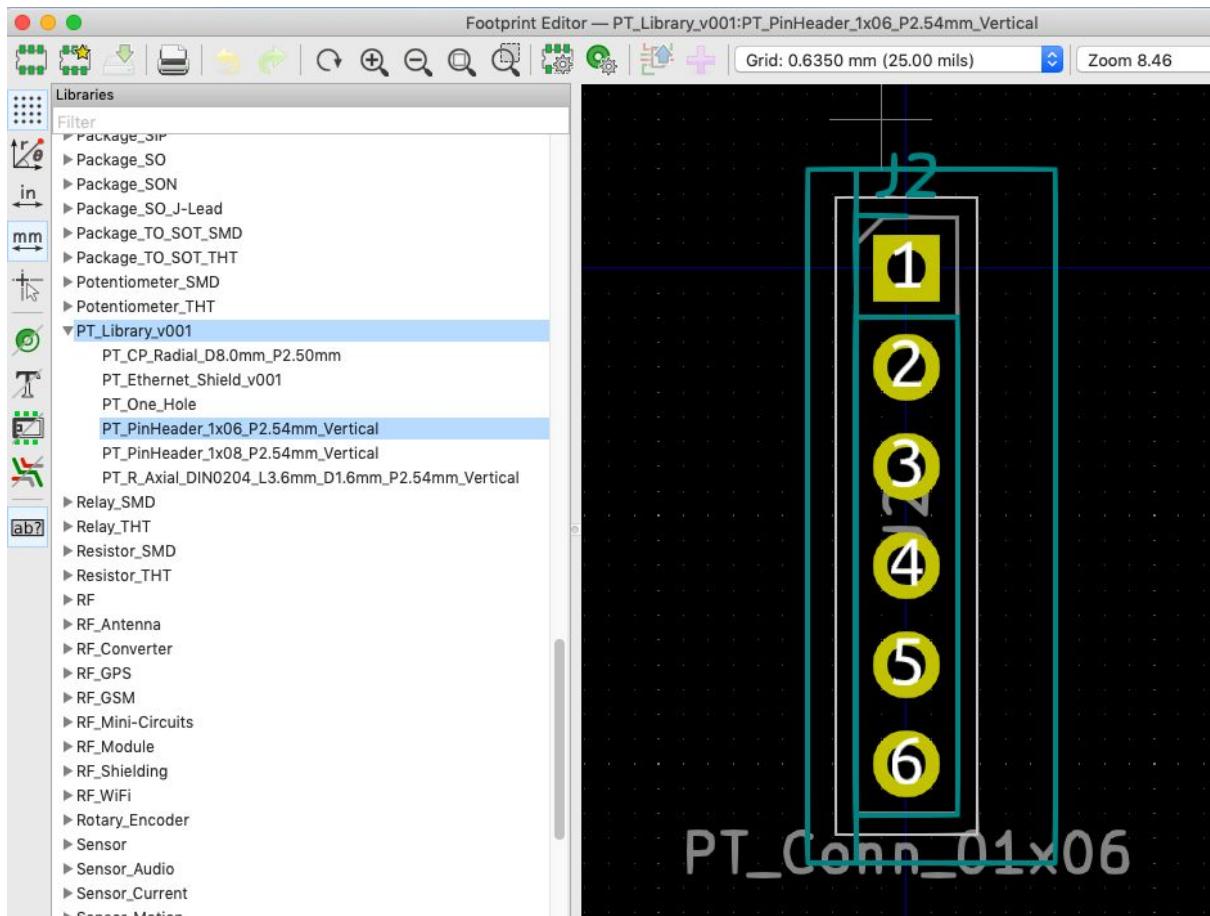
1. Click on *Preferences > Manage Footprint Libraries*.
2. Click on *Browse Libraries* and navigate to the downloaded .mod file.  
Then click OK. The library will appear in the *Global Libraries* tab.
3. In the table, make sure that the Plugin Type is set to *Legacy*. Then click OK.
4. Click on *Load footprint from library > Select by Browser*.
5. Navigate to the footprint you imported and double-click to open it.

Footprints can also be manually added. This was done for the PCB which sits atop both input and output Arduino's. Of a little interest was the spacing between the sets of pins was not the standard 0.1" (1.27mm/50mils), but was 0.5" (0.635mm/25mils). All pins had the same size. Its worth remembering that not all components have the same pin width, especially of note is power connectors. If adding Footprints save them to a common library and specify

its path as absolute not relative. Found I had referenced the library originally as relative, but then it wasn't found for new projects. The path to the library I used (PT\_Library\_001) is set here



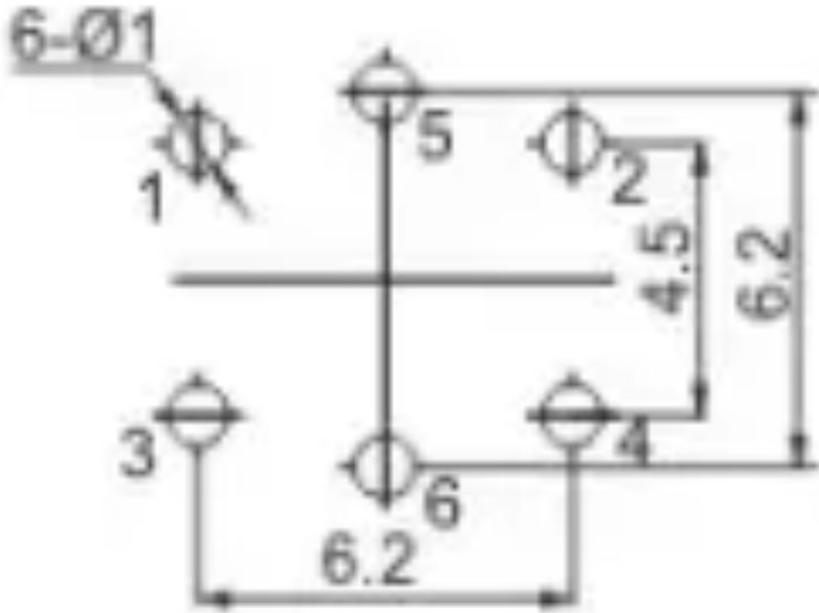
The shield does not have a boundary, this was intentional as otherwise alerts would be generated with the test routine. The following connector does have a silk screen boundary and a Court Yard (F.CrtYd) which helps ensure that components are not placed too close together. The silkscreen boundary is for the benefit of the user placing the component, the Court Yard is used by the Design Rule Checks to ensure adequate clearances have been given.



The tactile switches weren't defined in the library so footprints were added. Of note is the different pin sizes, the switch pins had a max width of 0.8mm, but the led pin width was 0.5mm.

#### Observed hole sizes from projects

Component	Inner Hole	Outer Pad
Small Resistor	0.762 mm	1.524 mm
Electrolytic	0.8 mm	1.6 mm
IDC Connector	1.0 mm	1.7 mm
Power Connector	1.52 mm	3.0 mm

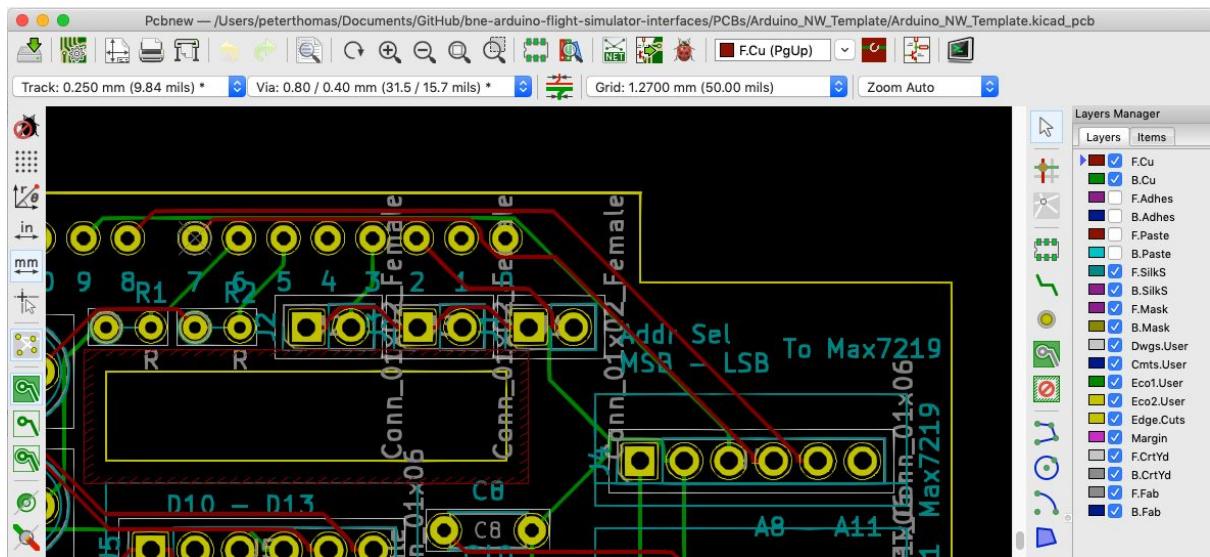


The outline is drawn using the line tool after the Front Silkscreen layer is selected on the right hand side.

Commonly used footprints can be saved in a separate library, which helps ensure consistency between projects. The Library I used was PT\_Library\_v001.

Once the circuit diagram is complete – save it and open up the PCB. One thing that takes a little getting used to is using the Keyboard and Mouse to do the editing, or a little more use of the Right Click. As an example to move something around you select with the mouse and then either press M or right-click Move.

If you need to add a hole, for example in the Ethernet Shield PCB, a slot was placed so Led status could be monitored, the hole/slot is drawn on the Edge.Cuts layer. A keepout area is added so the routing tool didn't place tracks too close to any edges.

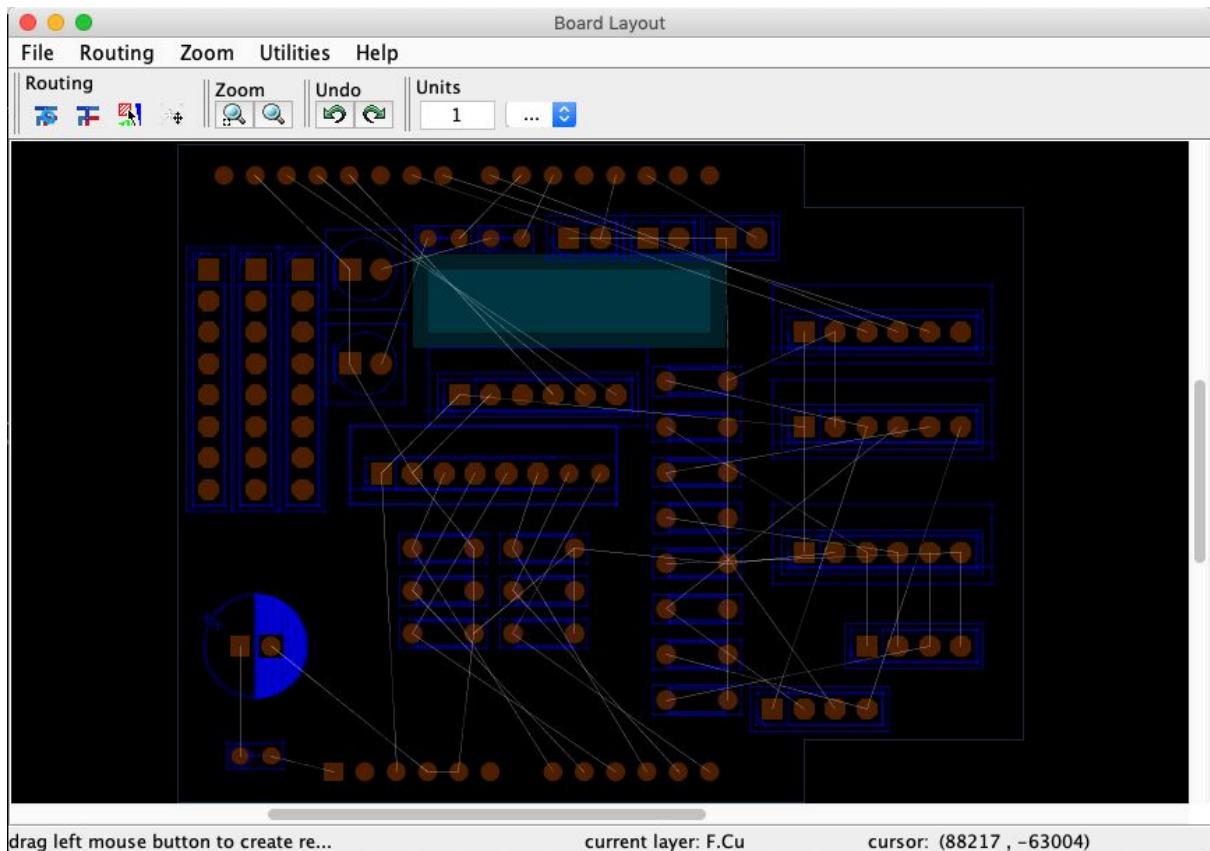


Once components are correctly positioned then you can either manually route tracks or use Free Router. Whilst the Freerouting page calls out that it is included in layout – you won't find it with a simple search when using the Mac – Right Click on Layout in Applications folder – show package contents<sup>4</sup> – and copy it to a folder that's easy to find.

If you'd like to completely restart the routing of all tracks – Edit → Global Deletions

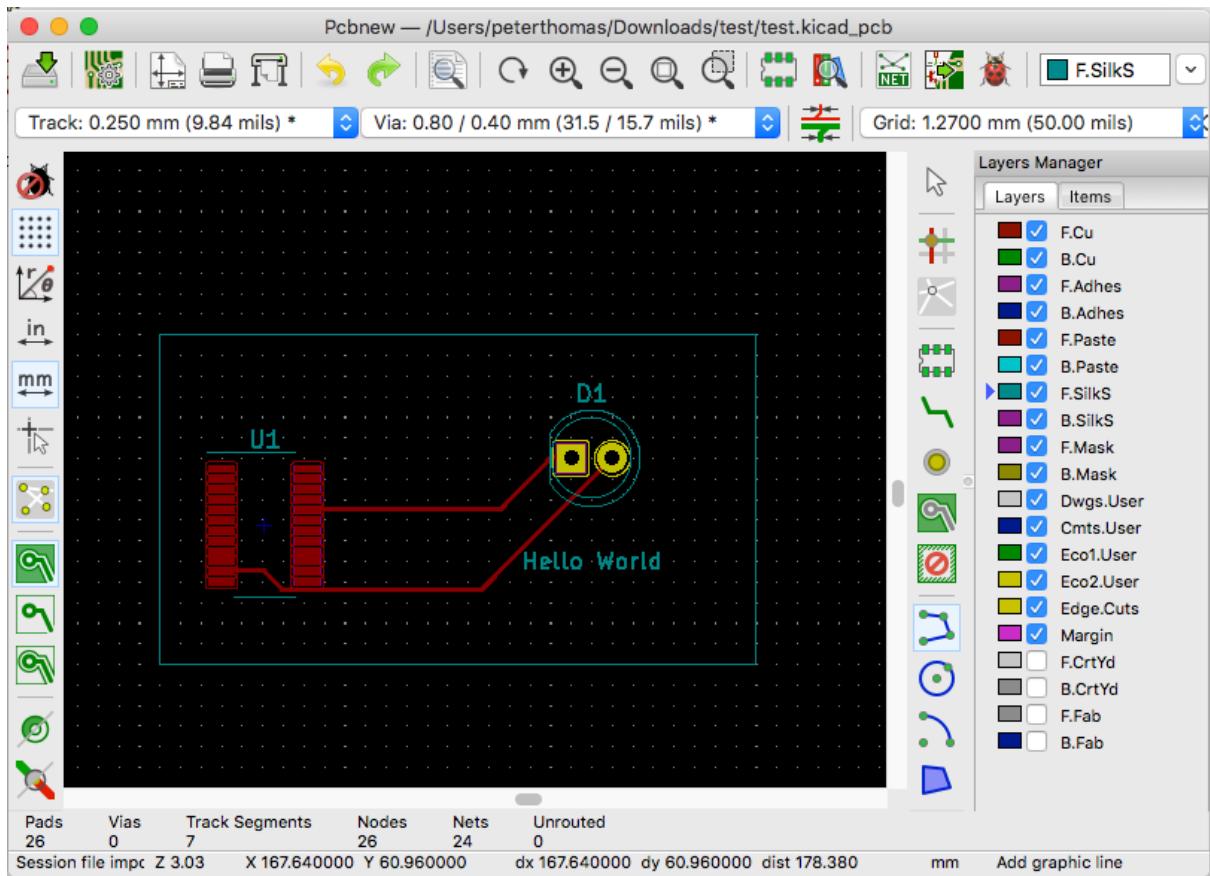
A dialog to the Freerouting tool will be opened. In this dialog you can export a Spectra Design File (.DSN). This is the input file you will need with FreeRouting. After opening FreeRouting open this file and perform the routing. When routing is finished store the result to a Spectra Session File (.SES). In the same KiCAD dialog this file with the routing results can be imported. I did find on one of my Macs the buttons did not display in the toolbar – the button of interest is in the top left.

<sup>4</sup> The version of freeRouter.jar I used was version 1.3.1 which was based on 1.2.43 from Alfons Wirtz It is 1.4M

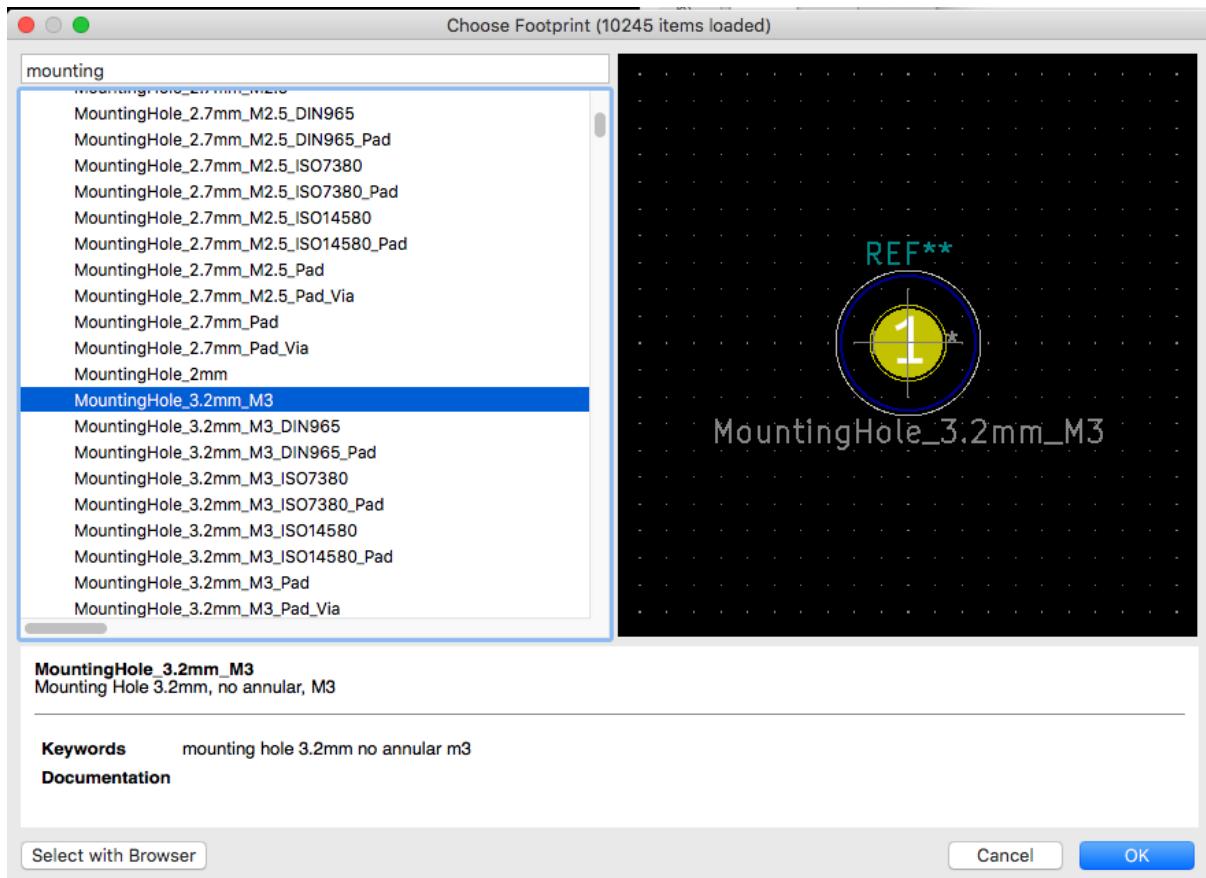


Then return to the PCB Editor in KiCad and import Spectra Session File.

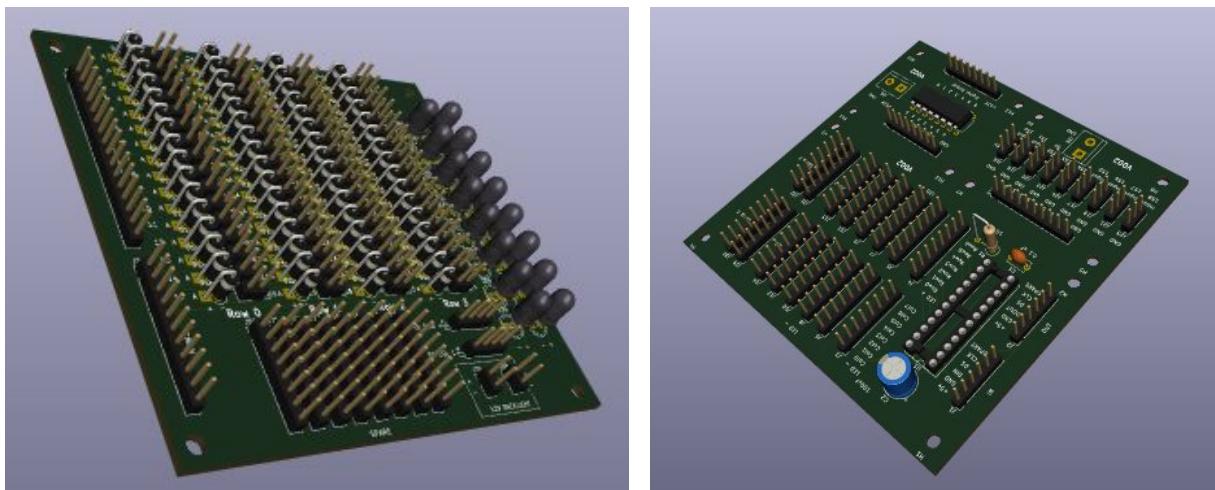
You can then add useful text to the silkscreen – don't forget to select the Front Silkscreen layer and potentially hide layers from the view – as it may look like values will be drawn on the PCB, but they aren't.



Earlier versions of KiCad didn't support mounting holes, they are supported today. Simply select add footprint. One thing you may do is assign a 'counter' to the hole otherwise you the plot to gerber files.



Once the PCB design is ready to send off – Plot the PCB and generate drill files (an option from the Plot Dialog box). It is probably a good idea to preview the PCB in 3D view



## Pre-Manufacturing Checklist

- Check component clearances around mounting holes (the bolt head and nuts are a bunch larger than the hole)
- Check component clearances between component (particularly headers)
- Check component pads are not interfering with PCB supports (eg the ones on the DDI)
- Check there is good clearance from PCB edges
- Check drill holes are large enough for larger components
- Where appropriate is text on the silkscreen to assist with assembly, monitoring and troubleshooting
- If larger amount of power are being carried, are the tracks wide enough

## Manufacturing the PCB

There are a bunch of companies who specialise in the manufacturing of Prototype PCBs. There are a couple of things to consider when selecting a manufacturer, firstly what's the lead time for the PCB, and what's the costs to ship the completed board.

I've recently started using PCBgogo<sup>5</sup>, they offer 2 day turn around with a solid web interface, which makes ordering, tracking, and reordering very simple. Additionally they offer an online gerber viewer so you can preview your files. Your files are checked before the order is accepted, so by the time you get to actually place the order – the files are verified, and shipping costs can be accurately estimated.

If the PCB is smaller than 100mm \* 100mm then 5 boards cost only USD\$5 to manufacture, plus the cost of shipping. PCBgogo offer a wide choice of courier companies – so if you are in a hurry you could have the PCB in your hands within 7 days of submitting the gerber files.

A couple of little tricks

- The minimum number of boards for an order is 5, which cost \$5, I've found I can order 10 boards and still only pay \$5 I'm guessing this is determined by the size of the board. So try it out :)
- Shipping is often more expensive than the boards, you can add multiple board designs to a single order.
- I did manage to confuse things a little by adding lines to the silk screen which PCBgogo thought were lines for cutting, so if you run into a speed bump like that just delete the lines from the silk screen. Normally designs are validated with 10 minutes – if it takes more than a couple of hours (during PCBgogo opening hours), something is up.

## PCBs Used in the Project

There are 7 PCBs used in the projects

5 <https://www.pcbgogo.com/>

	Version	Brief Description
Arduino Output	001	Attaches to output Arduino. 2 Servo outs, 1 Digital out, 1 Max7219 Out. Optional (but recommended) 5V input for driving Max7219. The servos are supply 5V at the Output PCB
Output	002	Basically three PCBs in one. The Max7219 Led driver, with multiple connections
Arduino Input	001	Attaches to the input Arduino, connectivity only aside from the 10K pull-up resistors
Input	001	Connectivity only with multiple connections for Panel PCB inputs to connect to.
Small Panel	002	For the smaller panels, supporting up to 24 switch positions and 8 LEDs. It also has backlighting connector and a row of pins for miscellaneous requirements.
Large Panel	002	For the larger panels, supporting up to 64 switch positions and 16 LEDS. It also has a backlighting connector and a row of pins for miscellaneous requirements.
Arduino_NW_Template	002	Sites on top of the Ethernet shield for both input and output Arduinos. It is used to determine Mac and IP Address as well as breakout Analog inputs

## Arduino Output

This PCB attaches to the Arduino providing output services. The same PCB is used in the centre, left and front sections.

It has two connectors for servo out (actually each connector is duplicated enabling four separate panels with servos to be supports. Additionally there is a connector to connect to the Max 7219 section of the Output board, and a digital output connector. As the software uses pins on another section of the Arduino Mega, a piece of aerial plumbing is needed to connect the Max7219 interface.

The Servo Connectors provide a total of 18 outputs to the servo board sections of the servo break out section of the Output board. Pin 10 of the servo output connectors is the ground connector. R2 and R4 would normally be a jumper wire, for reasons that elude me now the spot for a resistor was added, but as it is going to ground is it better not to have it there.

The Servos receive +5V via connector on the Servo Breakout Section of the Output board.

Mega Expansion Port Pin No	Output Pin Servo1	
3	1	
4	2	
5	3	
6	4	
7	5	
8	6	
9	7	
10	8	
11	9	
	10	GND (via R2)

Mega Expansion Port Pin No	Output Pin Servo2	
12	1	
13	2	
14	3	
15	4	
16	5	
17	6	
18	7	
19	8	
20	9	
	10	GND (via R4)

The Digital Out Connector is generally intended for non-LED loads that need additional driver circuitry such as Magnetically held switches, relays, and higher voltage (eg 12V/24V) lamps. As with the Servo out, pin 10 on the connector should be wired to ground, ie a jumper installed for R3. As the UN2003 only has 7 inputs, typically an eight pin connector is installed in the last 8 pin positions of the Digital Out Connector

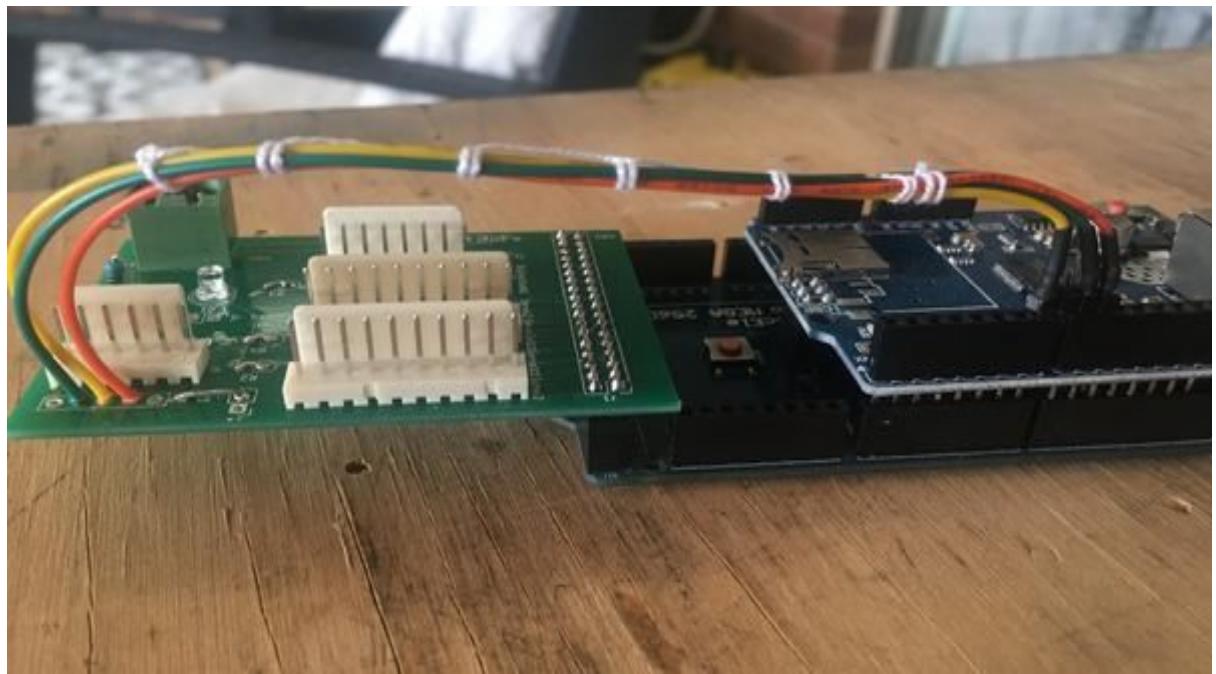
Mega Expansion Port Pin No	Output Pin Digital Out	

21	1	
22	2	
23	3	
24	4	
25	5	
26	6	
27	7	
28	8	
28	9	
	10	GND (via R3)

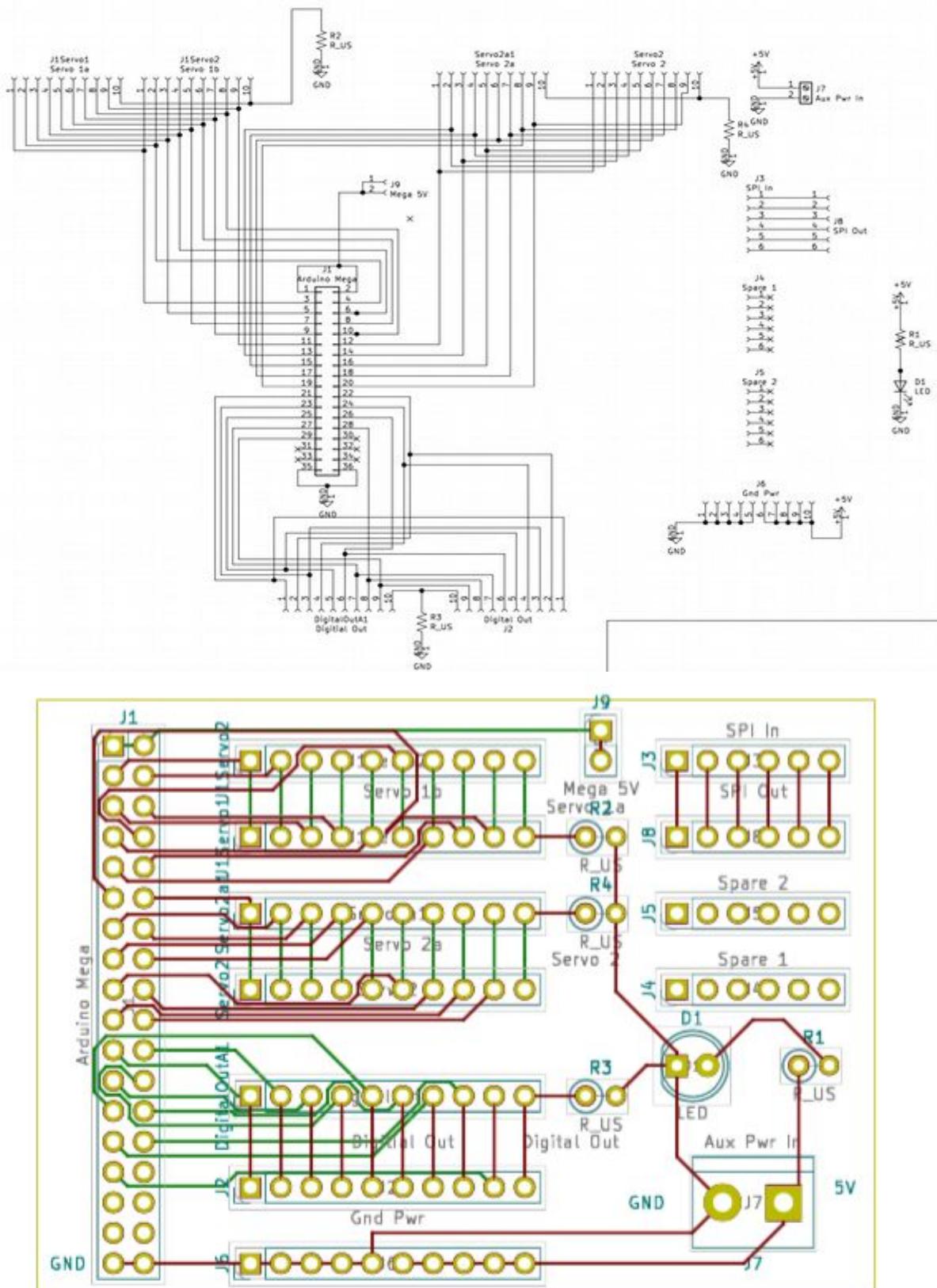
### Max7219 Port

Ports J3 and J8 are basically an anchor point for MAX7219 connectivity. It is optional to power the Max7219 through the Arduino 5V rail or from the 5V connector on this board. All of the pins for these connectors must be manually wired, including GND, 5V, and the aerial connections to the pins on the Mega. The pinout used is based on that commonly found with Max7219 mini-circuit boards found on Ebay, which are useful for performing tests.

Pin	Function	Cable Colour
1	+5V	Red
2	GND	Black
3	Mega Pin 9	Orange
4	Mega Pin 7	Yellow
5	Mega Pin 8	Green
6	Not Used	



The drawing was done before I understood the benefit of using buses, so it is a little busy.

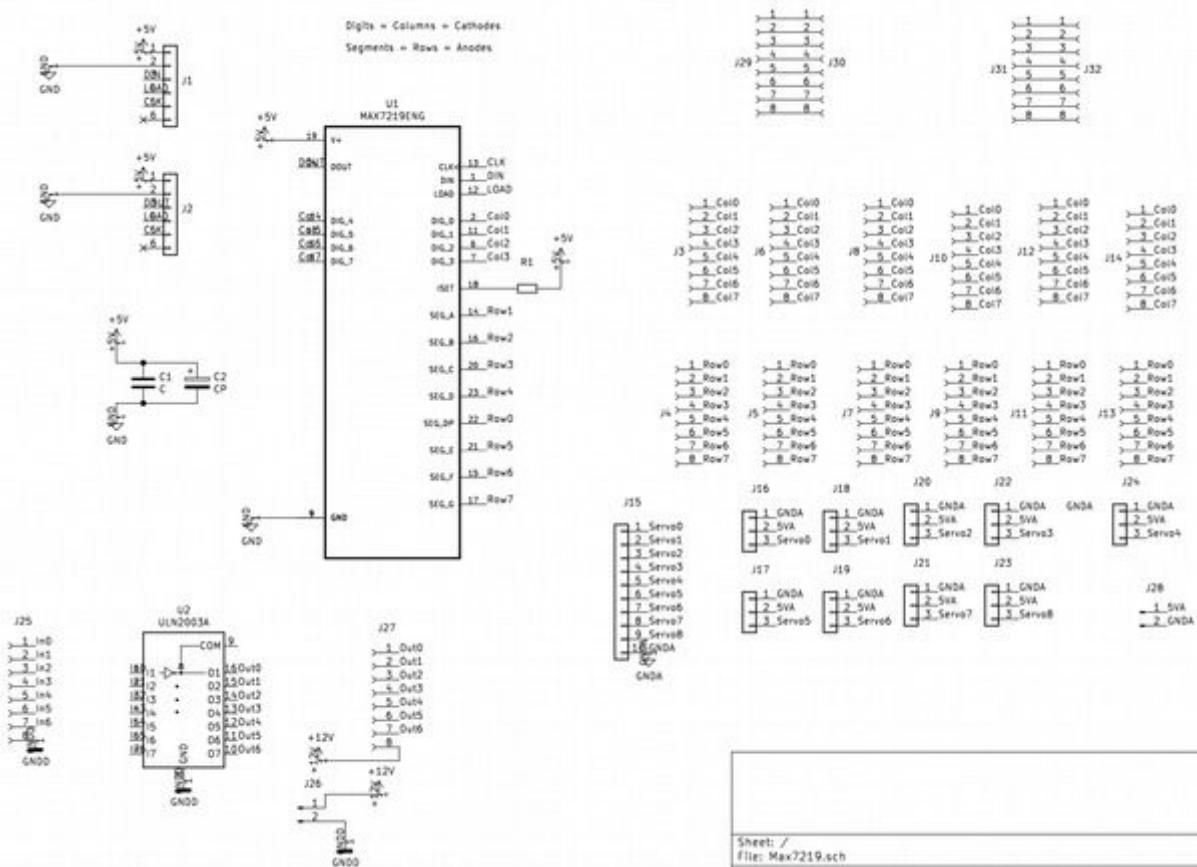


# Output PCB

This PCB is really a combination of three PCBs, a Max7219, a servo breakout (passive), and a ULN2003 driver for non-LED loads. Note the three PCB sections are completely isolated and can be physically separated if needed. Its probably a smart idea to leave the PCBs together, even if you aren't planning to use pieces of the board. You can simply leave the components out of that section.

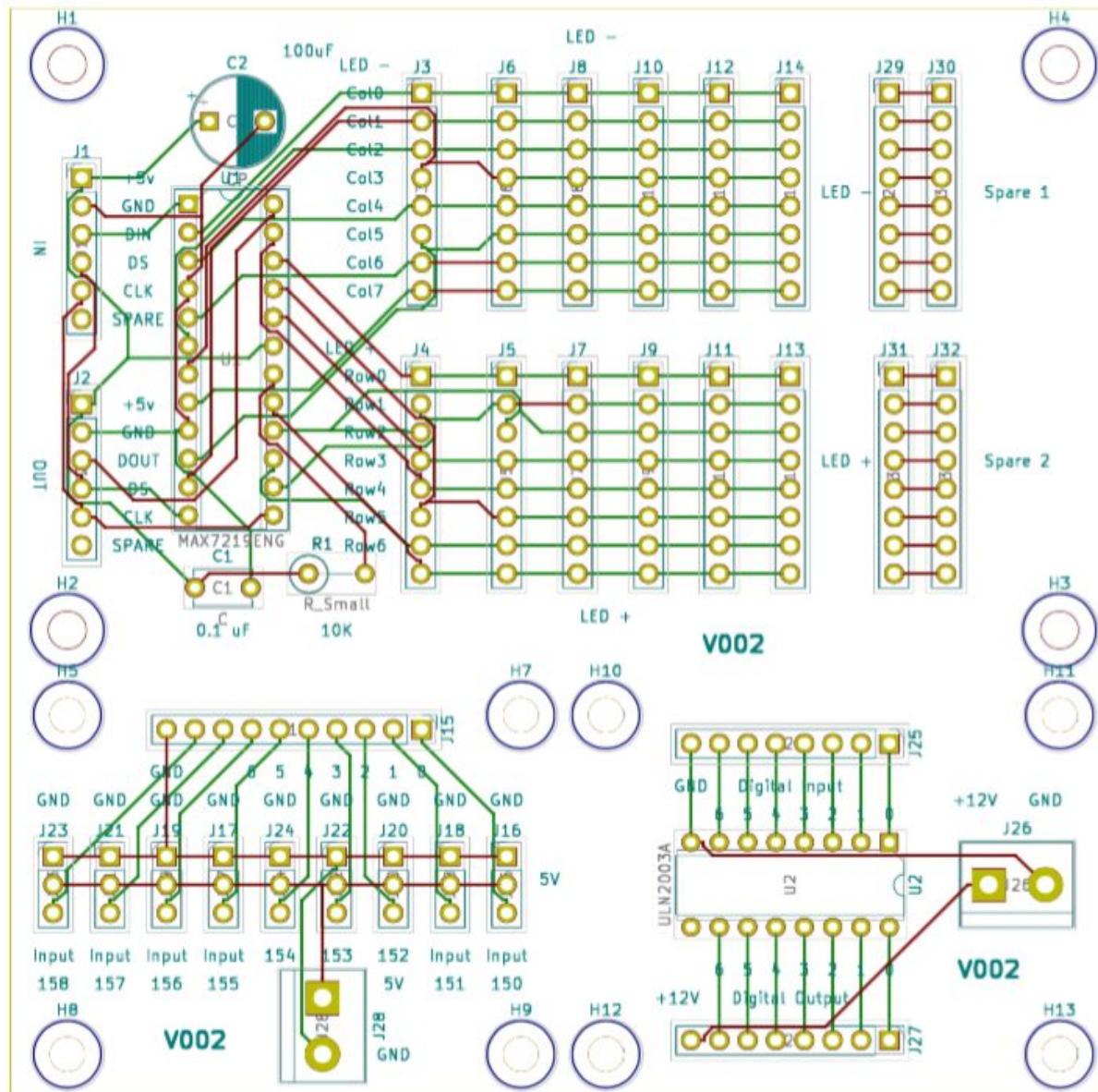
Note that J2 - Data Out hasn't yet been tested.

Pin – J1 Data In	Function
1	+5V
2	Gnd
3	Data In - DIN
4	Load
5	Clock - CLK
6	Not Used



Note there are three different grounds, GNDA, GND0, and GND. This was done to enable

the board to be split in three if needed, but ultimately these represent the same electrical ground.



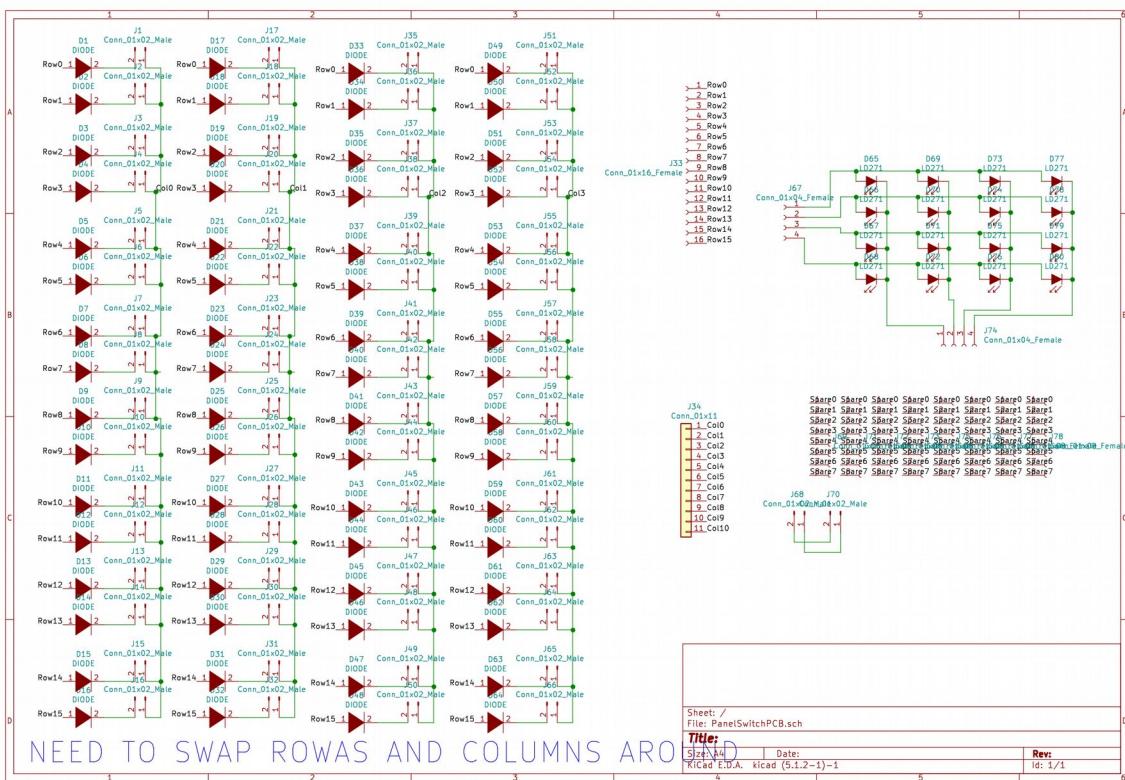
Components used in Output PCB

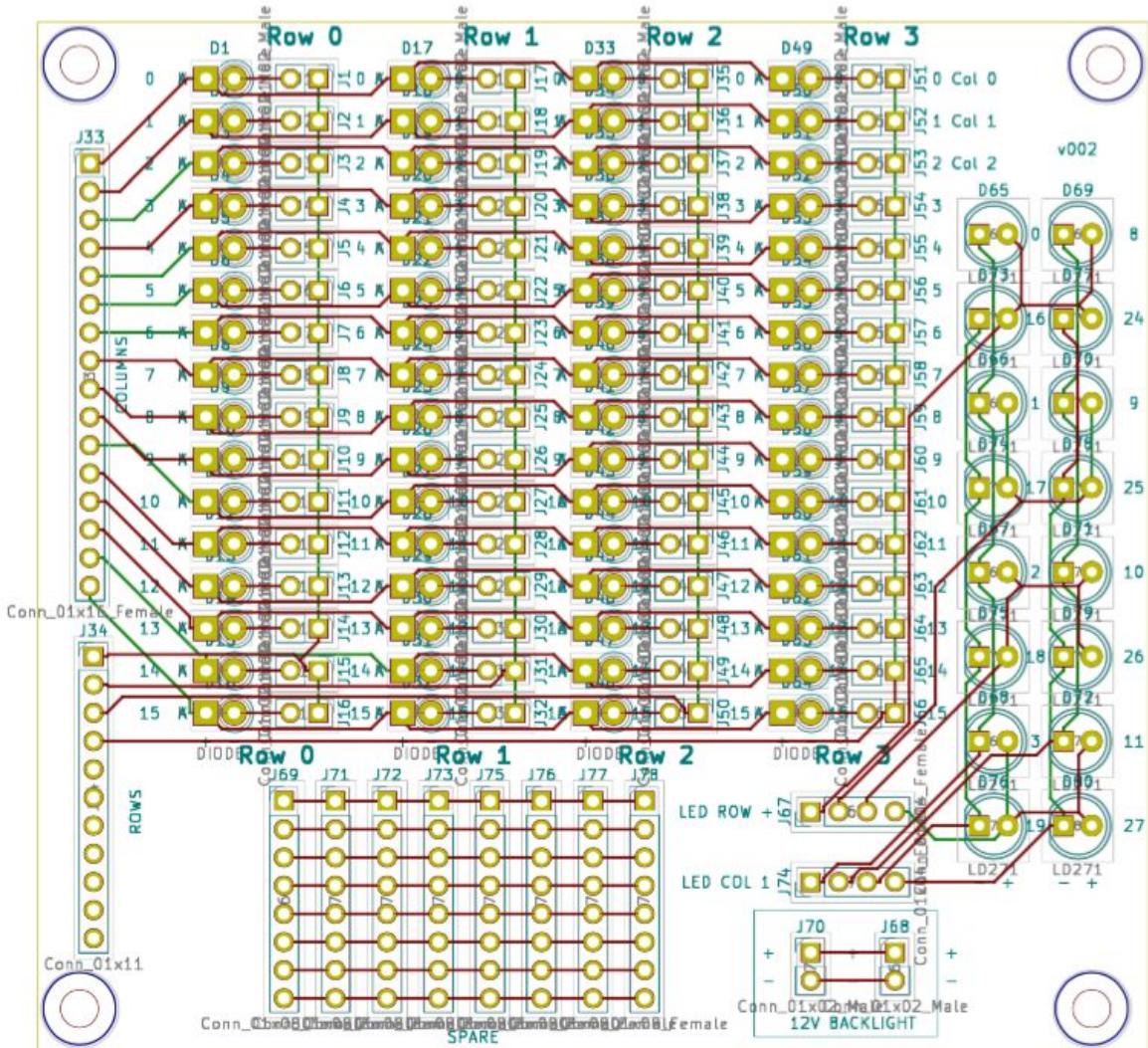
Ref	Value	Description
U1	Max7219	Led Driver for driving up to 64 Leds or Eight Seven Segment displays
U2	UN2003A	Array of seven NPN Darlington transistors capable of 500 mA, 50 V output.
C1	01.uF	Capacitor for reducing high frequency noise
C2	100-300uF	Electrolytic Capacitor for reducing lower frequency noise on the 5V rail.

Ref	Value	Description
R1	10K	Sets the current per LED of the MAX7219

## Panel Switch PCB

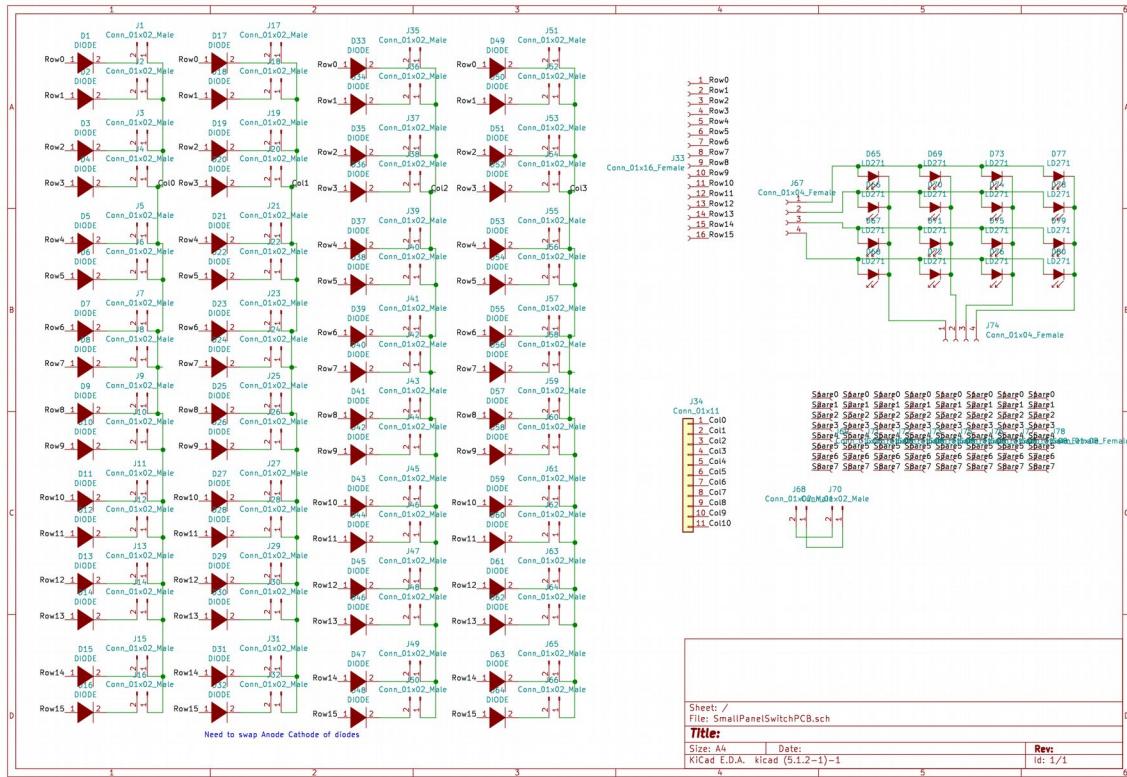
The Panel Switch PCB is designed to provide an interface to a large panel, or perhaps a bunch of panels that are collocated. Supporting up to 64 switch positions, along with 16 LEDs, it should support the largest of panels. Both the Panel Switch PCB and the Small Panel Switch PCB support the diodes needed for a Matrix input. Single switches could use connectors (eg J1-J64), but is also practical. The diode cathode is in the round shapes outline in the PCB

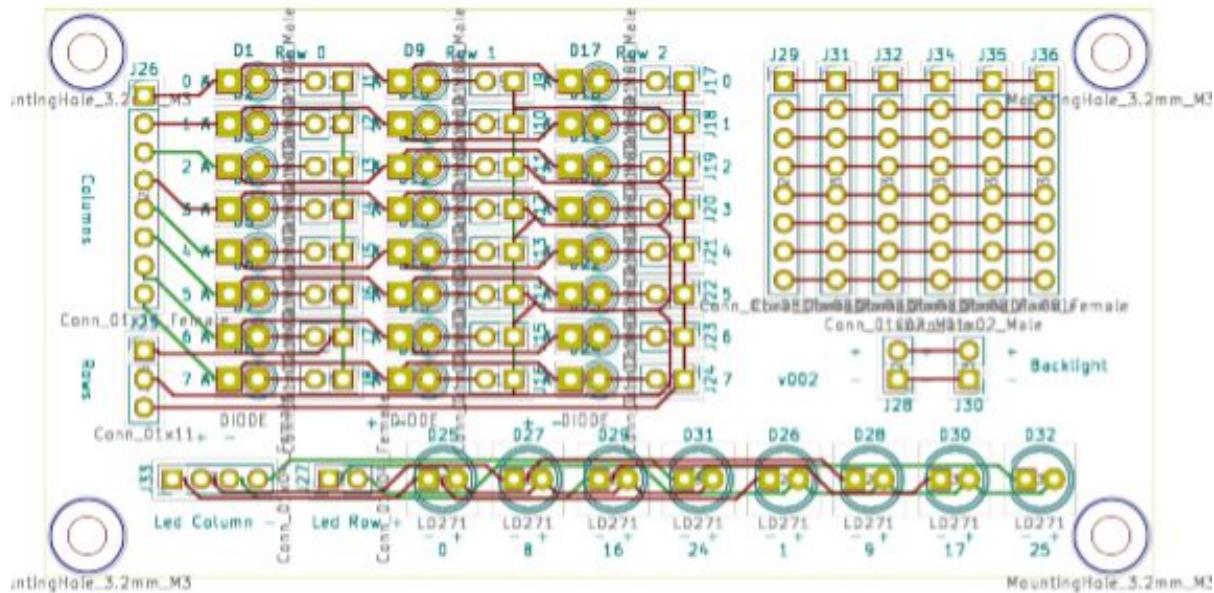




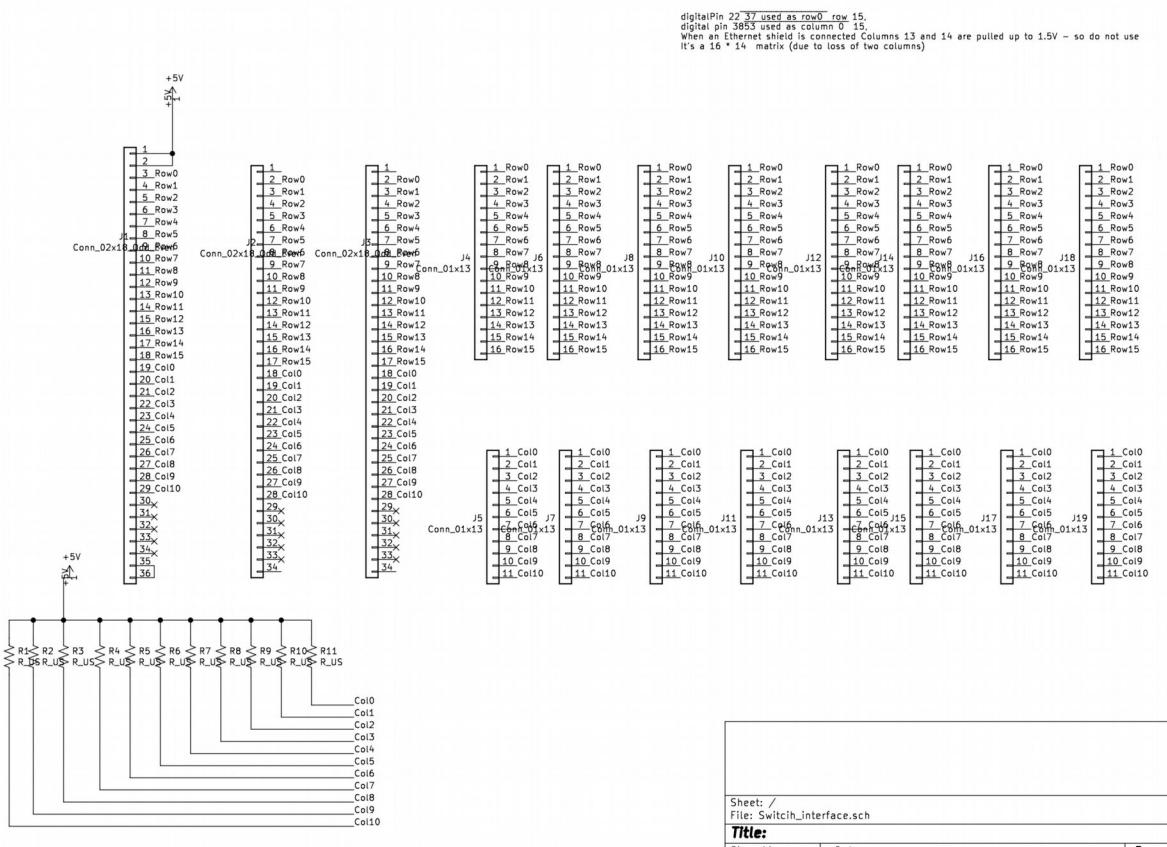
# Small PCB

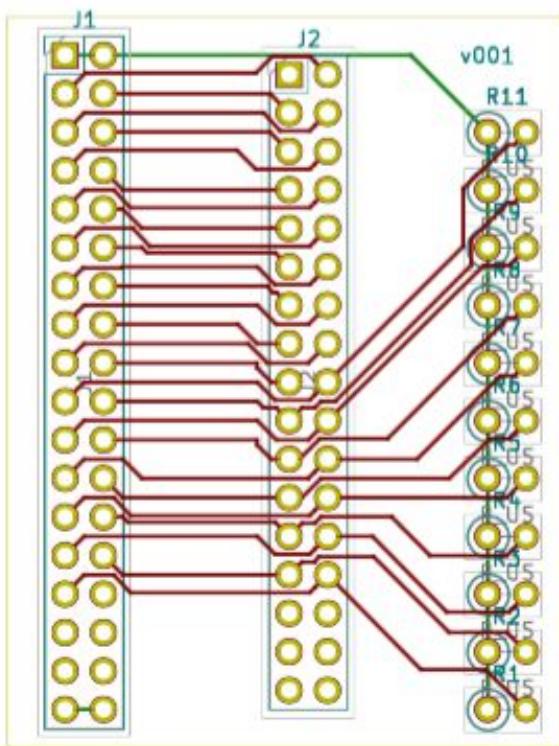
NEED TO CORRECT THE DIODES IN THE DIAGRAM THE ROWS ARE PULLED DOWN – SO THE DIODE CATHODE SHOULD FACE TOWARD THE ROW. COLUMNS ARE PULLED UP BY RESISTORS

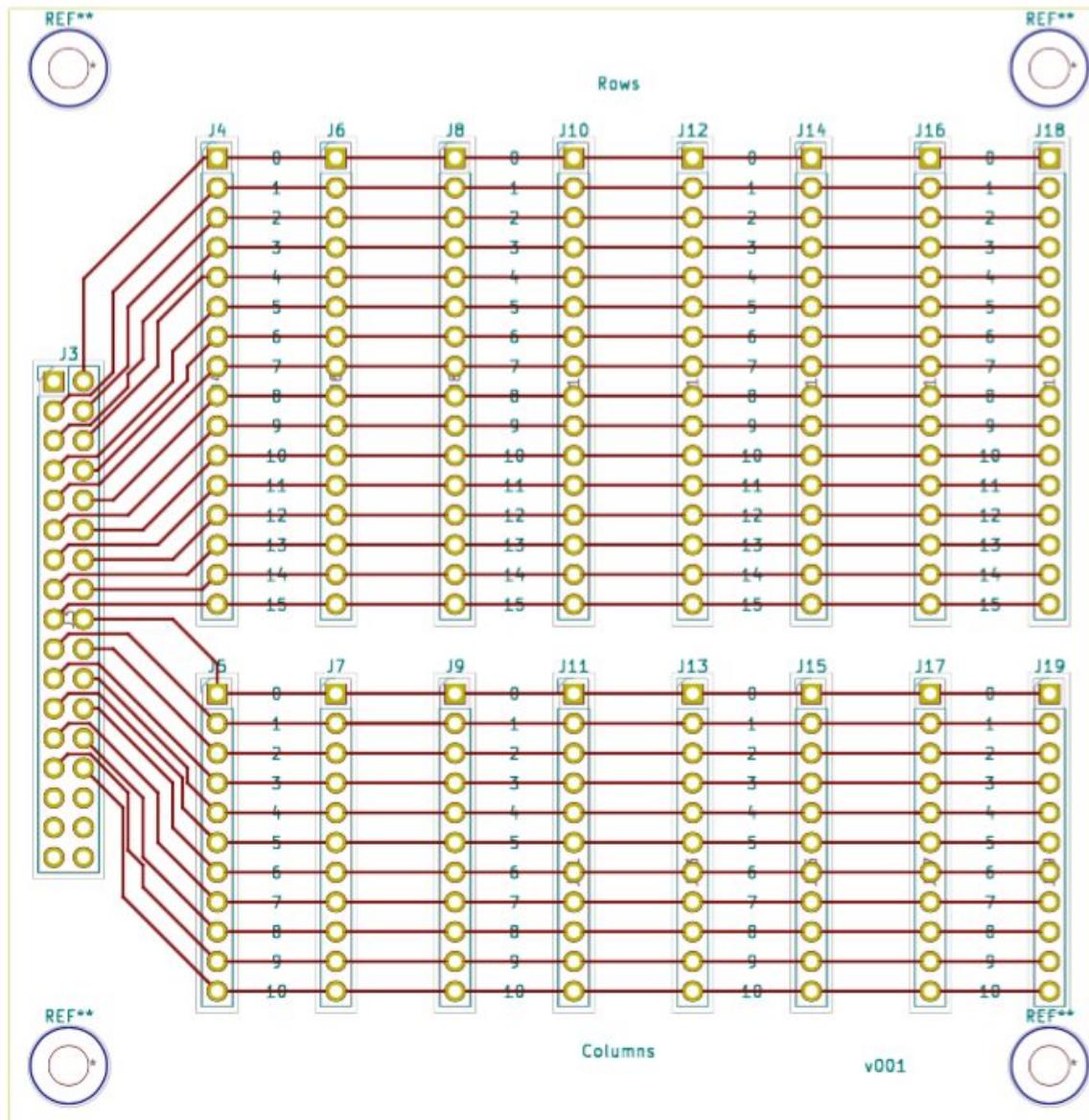




## Arduino Digital Connector







## Arduino\_NW\_Template

This PCB was a late addition. It is used to reduce the number of different versions of code by dynamic determining Mac and IP Address based on jumper settings by D3 to D7. With no jumpers wired it assumes the identity of Device 0. There still needs to be a separate lookup table for Input and Output devices. J1-3 are the address selection - J4 is the Data out for the Max7219 cards.

Input Addresses range from 172.16.1.10 → 172.16.1.17, and output devices range from 172.16.1.20 → 172.16.1.27. If non standard inputs and outputs are needed, the inputs should come from 172.16.30 → 172.16.1.39 and outputs 172.16.1.40 → 172.16.1.49.

The slot towards the Data output side allows the Ethernet Status LEDs to still be seen when the shield is installed.

When the Arduino initialises both LEDs are turned on and then turned off. Once the Board ID is determined the Status led flashes a number of times to reflect the board ID.

The Ethernet LED is turned on until a packet is able to be sent to the Pi (172.16.1.2), in reality – it stays on until the ARP of 172.16.1.2 is able to be determined, but what it does do is prevent the W5100 shield from sending the UDP packet within the timeout period.

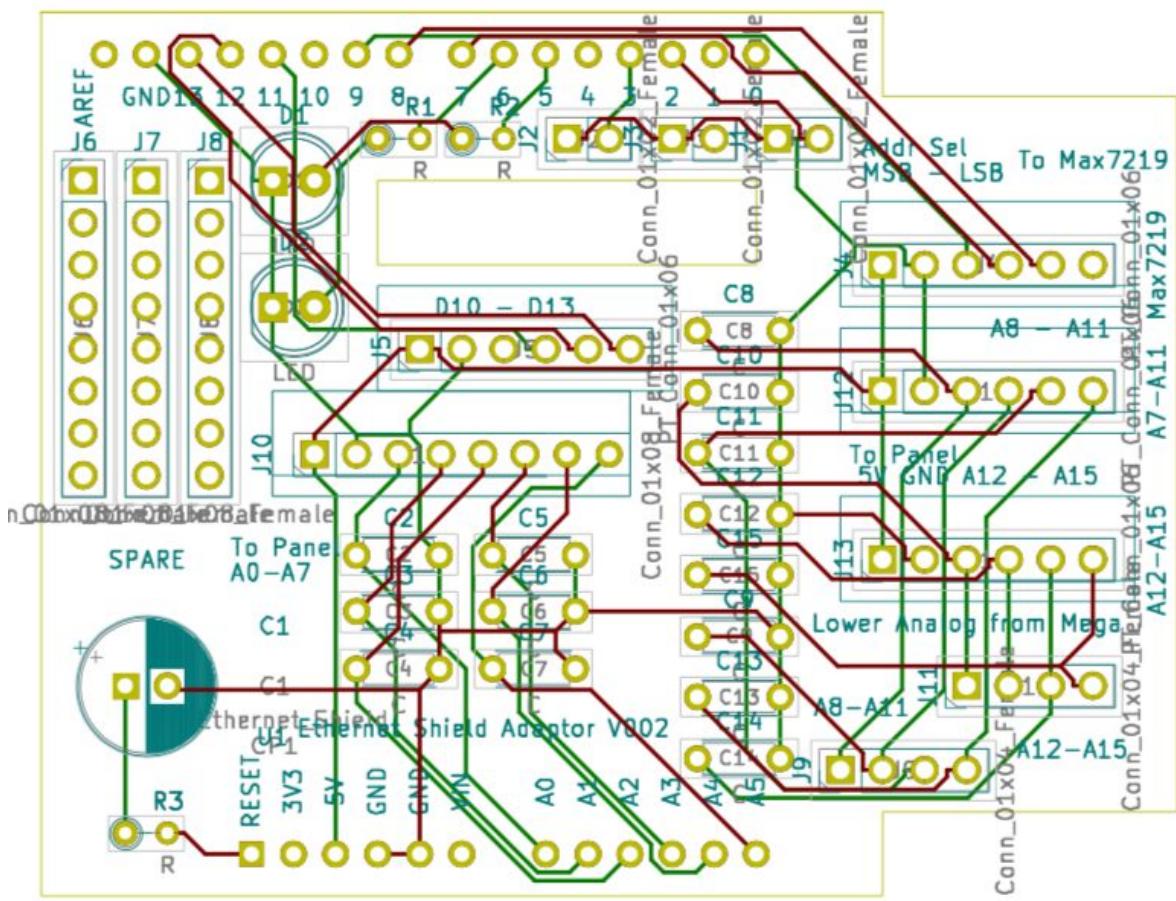
Once the packet is able to be sent the Pi, the status led flashes once a second.

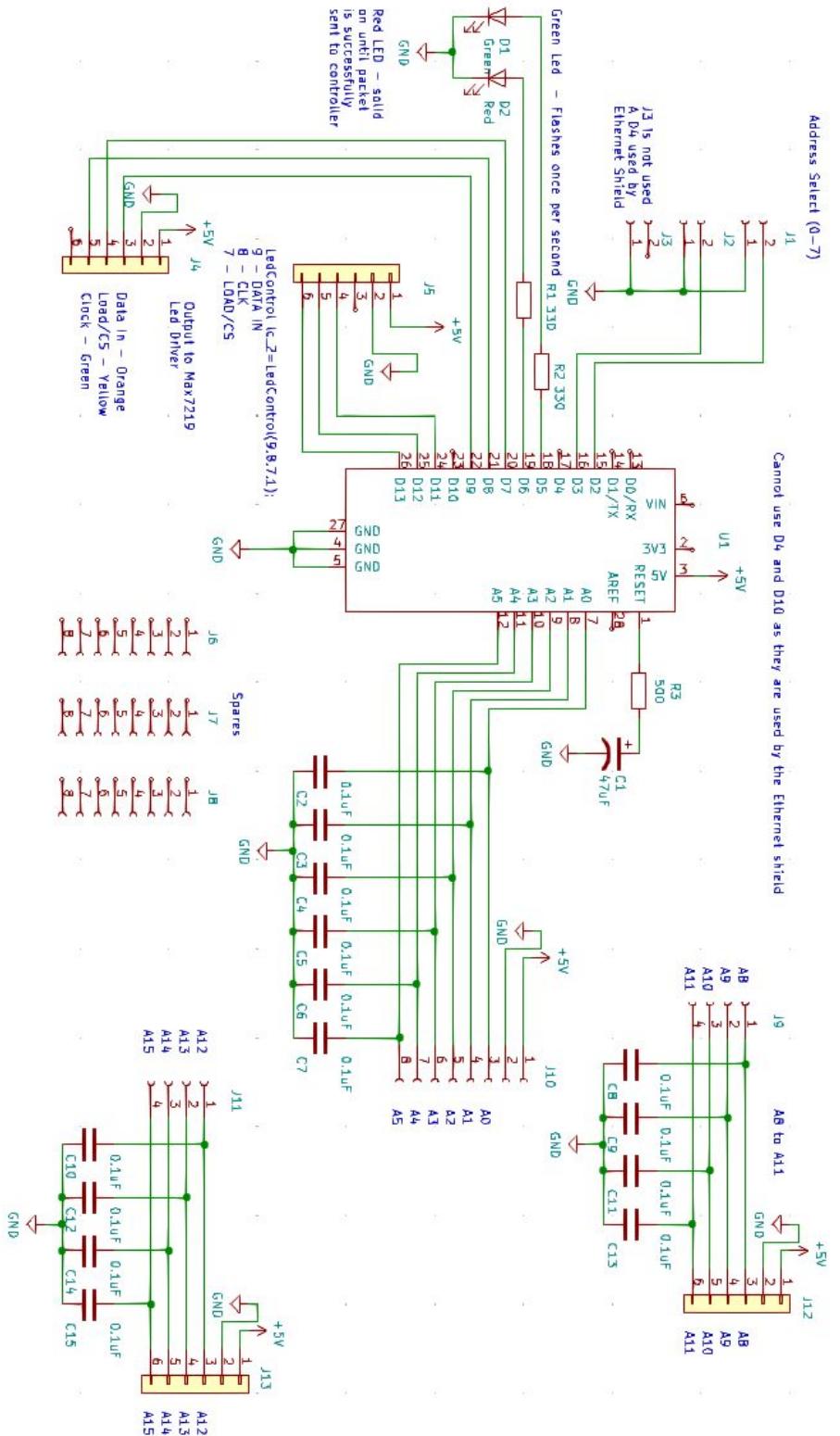
Every packet is prefix by a D followed by a two digit number. All values are comma separated. Digital Values are always prefixed by a 3 digit number (eg D087), analog values are always prefixed by an A followed by a two digit number (eg A13).

Analog and digital state is not sent during initialisation, to prevent unwanted disruption of the Sim, and also the Sim (well Pi) may not yet be operational.

All inputs (digital and analog) are sent in response to a CQ

Component ID	Value	Description
C1	47uF	Reset hold timer RC network with R3 for W5100
C2, C3, C4, C5, C6, C7, C8, C9, C10, C11, C12, C13, C14, C15	0.1uF	Analog input noise reduction
D1	Green LED	Shield Status LED – On start up flashes once along with Network Status LED and then flashes the Device ID (eg device 02 flashes 2 times). Flashes once per second when module active.
D2	Red LED	Network Status LED – On start up flashes once along with Shield Status LED, and then remains solid on until a packet can be sent to the master controller.
R1,R2	330	
R3	500	Reset hold timer RC network with C1 for W5100





# Testing the PCBs

## Output Testing

UDP\_Test\_Sender can be used to test the outputs. It defaults to the IP Address of the Front Arduino (172.16.1.21) with port 13135.

UDP\_Test\_Sender - `ctrl-c` at anytime to exit

```
Enter Target IP Address [172.16.1.21]:  
Enter Target Port [13135]:  
Would you like Run LEDs Test [N]:  
Would you like Run Digital Outputs Test [N]:  
Would you like Run Servos Test [N]:  
Would you like all commands to be prefix with a D [Y]:  
2019-09-01 07:35:22,645:DEBUG:UDP target IP: 172.16.1.21 UDP target Port: 13135  
Enter Command String to Send: 0:1  
2019-09-01 07:35:28,709:DEBUG:UDP target IP: 172.16.1.21 UDP target port: 13135  
2019-09-01 07:35:28,794:DEBUG:Sending: "D,0:1"  
"  
Enter Command String to Send:
```

---

The LED test turns all LEDs on, turns them off, and then walks all of the LEDs.

The Digital Output is similar to the LED test, but only does the last 7 Outputs. Whilst there are 9 Outputs allocated on the PCB, it is expected that they all won't be needed, and as the UN2003 only has 7 Darlington Pairs it made sense not to confuse the operator by nothing being displayed when Port 0 and 1 of the digital outputs are toggled. As with the LED test all ports are turned off, turned off and then walked.

The Servo Test first prompts the operator with which port on the Arduino Output PCB is being tested (first port maps to the outer connector, the second maps to the inner connector), and then for each port swings the servo towards the minimum (but not the absolute minimum), then a rough centre, then towards the maximum (again the not absolute maximum), then back towards the minimum and then moves to the next servo. There are 9 servos on each connector.

```

UDP_Test_Sender - ctl-c at anytime to exit

Enter Target IP Address [172.16.1.21]:
Enter Target Port [13135]:
Would you like Run LEDs Test [N]:
Would you like Run Digital Outputs Test [N]:
Would you like Run Servos Test [N]: Y
2019-09-01 07:25:35,508:DEBUG:UDP target IP: 172.16.1.21 UDP target Port: 13135
Testing first or second connector? [1] or [2] defaults to [1]: 1
Lower Port 0 J16 - Press Enter to continue...
2019-09-01 07:25:48,683:DEBUG:UDP target IP: 172.16.1.21 UDP target port: 13135
2019-09-01 07:25:48,746:DEBUG:Sending: "D,150:40
"
Mid Port 0 J16 - Press Enter to continue...
2019-09-01 07:25:50,475:DEBUG:UDP target IP: 172.16.1.21 UDP target port: 13135
2019-09-01 07:25:50,548:DEBUG:Sending: "D,150:100
"
Upper Port 0 J16 - Press Enter to continue...
2019-09-01 07:25:51,114:DEBUG:UDP target IP: 172.16.1.21 UDP target port: 13135
2019-09-01 07:25:51,169:DEBUG:Sending: "D,150:160
"
Lower Port 0 J16 - Press Enter to continue...
2019-09-01 07:25:55,938:DEBUG:UDP target IP: 172.16.1.21 UDP target port: 13135
2019-09-01 07:25:56,005:DEBUG:Sending: "D,150:40
"
Moving to Next Servo

```

---

If none of the block tests (i.e. LED, Digital Outputs, Servos) are selected, then the operator is asked if commands should be prefixed by a 'D'. This typically will be used as the protocol for outputs expects the first character in the packets to be a 'D'. The command is then interactively entered, and when enter is pressed, the command prefixed by the 'D' and terminated with a 0x0A (chr(10)) is sent.

## Input Testing

The Arduino Input controllers send input updates to both the input process (26027 – PyHWLink\_Pri\_Node\_Input.py) and the reflector (27000 – UDP\_Reflector.py) on the Raspberry Pi (172.16.1.2). As the UDP\_Reflector doesn't use any configuration files as such it is the simplest to use for testing. For testing I use the wired port of my Mac Mini with its address statically assigned to the same as the Pi (172.16.1.2), and run UDP\_Reflector.py.

Port Reference from udp\_input\_controller.ino

```

// Raspberry Pi is Target
IPAddress targetIP(172,16,1,2);

const unsigned int localport = 7788;
const unsigned int remoteport = 26027;
const unsigned int reflectorport = 27000;

```

```
2019-09-01 08:24:24,244:INFO:Unable to find UDP_Reflector_config.py
2019-09-01 08:24:24,260:DEBUG:Checking Command Line parameters
2019-09-01 08:24:24,293:DEBUG:options:{'opt_W_Host': None, 'opt_W_Port': None}
2019-09-01 08:24:24,326:DEBUG:arguments:[]
Listening on port 27000
2019-09-01 08:24:27,361:DEBUG:From: 172.16.1.12 7788
2019-09-01 08:24:27,377:DEBUG:Message: b'D02:000:0'
2019-09-01 08:24:27,411:DEBUG:Processing UDP String
2019-09-01 08:24:27,448:DEBUG:From: 172.16.1.12 7788
2019-09-01 08:24:27,477:DEBUG:Payload: b'D02:000:0'
2019-09-01 08:24:27,532:DEBUG:Iterations since last packet 2150
2019-09-01 08:24:27,802:DEBUG:From: 172.16.1.12 7788
2019-09-01 08:24:27,811:DEBUG:Message: b'D02:000:1'
2019-09-01 08:24:27,844:DEBUG:Processing UDP String
2019-09-01 08:24:27,878:DEBUG:From: 172.16.1.12 7788
2019-09-01 08:24:27,911:DEBUG:Payload: b'D02:000:1'
2019-09-01 08:24:27,944:DEBUG:Iterations since last packet 174
2019-09-01 08:24:29,128:DEBUG:From: 172.16.1.12 7788
2019-09-01 08:24:29,145:DEBUG:Message: b'D02:001:0'
2019-09-01 08:24:29,179:DEBUG:Processing UDP String
2019-09-01 08:24:29,212:DEBUG:From: 172.16.1.12 7788
2019-09-01 08:24:29,245:DEBUG:Payload: b'D02:001:0'
2019-09-01 08:24:29,279:DEBUG:Iterations since last packet 833
Keepalive Sun Sep 1 08:24:29 2019
```

Figure 1: Output From UDP\_Reflector.py - Input Test

In the above figure we can see the first port from the right hand input controller (02) close (which goes to a 0 as the input is pulled from high to low) and then released. There is a bounce as it was closed by hand.

## Interfacing to the Sims.

### P3d

Traditionally I'd worked with FSUIPC, work has provided a very consistent interface through the different generations of Microsoft's, and now P3d sims. As SimConnect is now increasingly commonly used using that for the interface to the Sim.

This does mean at least part of the workload has to run on the PC. It is intended this will be a very shallow shim, with the bulk of the workload running on the Pi.

SimConnect is used to subscribe to a dataflow - which means that code must either be installed either on the PC running P3d or on a second PC with SimConnect installed and configured to point to primary PC.

As the code is light weight the plan is to run on Primary PC

Where necessary keystrokes will be used to send commands to the Sim (or using SimConnect)

Useful URLs

[https://www.prepar3d.com/SDKv4/sdk/simconnect\\_api/managed\\_simconnect\\_projects.html](https://www.prepar3d.com/SDKv4/sdk/simconnect_api/managed_simconnect_projects.html)

Variables

[https://www.prepar3d.com/SDKv4/sdk/references/variables/simulation\\_variables.html](https://www.prepar3d.com/SDKv4/sdk/references/variables/simulation_variables.html)

Code Fragments to Support SimConnect

```
// User-defined win32 event

const int WM_USER_SIMCONNECT = 0x0402;

// SimConnect object
SimConnect simconnect = null;

// this is how you declare a data structure so that
// simconnect knows how to fill it/read it.
// When Adding variables to receive need to add them to this datastructure as
// well as the request itself initDataRequest

[StructLayout(LayoutKind.Sequential, CharSet = CharSet.Ansi, Pack = 1)]
struct Struct1
{
    // this is how you declare a fixed size string
    [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 256)]
    public String title;
    public double latitude;
    public double longitude;
    public double altitude;
    public double airspeed;
    public double elapsedsimtime;
    public double zulu_time;
    public Int32 time_zone_offset;
    public double absolute_time;
    public double plane_heading_degrees_true;
    public double plane_heading_degrees_magnetic;
};

void simconnect_OnRecvSimobjectData(SimConnect sender,
SIMCONNECT_RECV_SIMOBJECT_DATA      data)
{

    displayText("Received something don't know what to do with it but trying");

    switch ((DATA_REQUESTS)data.dwRequestID)
    {
        case DATA_REQUESTS.REQUEST_1:
            Struct1 s1 = (Struct1)data.dwData[0];

            displayText("titles:           " + s1.title);
            displayText("Lat:             " + s1.latitude);
            displayText("Lon:             " + s1.longitude);
            displayText("Alt:             " + s1.altitude);
            displayText("Airspeed         " + s1.airspeed);
            displayText("Sim Time         " + s1.elapsedsimtime);
    }
}
```

```

        displayText("Zulu Time      " + s1.zulu_time);
        displayText("Time Zone Offset  " + s1.time_zone_offset);
        displayText("Absolute Time    " + s1.absolute_time);
        displayText("Plane Heading True " + s1.plane_heading_degrees_true);
        displayText("Plane Heading Mag   " +
s1.plane_heading_degrees_magnetic);

        UDP_Playload = "latitude:" + s1.latitude;
        UDP_Playload = UDP_Playload + ",longitude:" +
s1.longitude.ToString();
        UDP_Playload = UDP_Playload + ",altitude:" + s1.altitude.ToString();
        UDP_Playload = UDP_Playload + ",airspeed:" + s1.airspeed.ToString();
        UDP_Playload = UDP_Playload + ",zulutime:" + s1.zulu_time.ToString();
        UDP_Playload = UDP_Playload + ",timezoneoffset:" +
s1.time_zone_offset.ToString();
        UDP_Playload = UDP_Playload + ",trueheading:" +
s1.plane_heading_degrees_true.ToString();
        UDP_Playload = UDP_Playload + ",magheading:" +
s1.plane_heading_degrees_magnetic.ToString();

        span = DateTime.Now - TimeLastPacketSent;
        mS = (int)span.TotalMilliseconds;
        displayText("Its been this many mS since sending last packet: " +
mS.ToString()));

        if (mS >= 500)
        {
            Byte[] senddata = Encoding.ASCII.GetBytes(UDP_Playload);
            udpClient.Send(senddata, senddata.Length);

            TimeLastPacketSent = DateTime.Now;
        }
        break;

    default:
        displayText("Unknown request ID: " + data.dwRequestID);
        break;
}

```

# DCS

The type of interface used in DCS depends heavily on the aircraft being flown. The study models will use commands sent over UDP.

For the more generic models keystrokes will need to be send to the Sim. This means a shallow shim will need to run on the primary Sim. Current plan is to send Windows Keycodes over UDP, with the Windows API being used to inject keystrokes into the Keyboard buffer.

Testing basic LUA – turning off left most Stability switch in the A10C

```
Enter Target IP Address [192.168.1.138]: 172.16.1.3
Enter Target Port [7790]: 7780
Would you like all commands to be prefex with a D [N]:
2019-05-25 11:59:50,907:DEBUG:UDP target IP: 172.16.1.3  UDP target Port: 7780
Enter Command String to Send: C38,3003,0.0
2019-05-25 12:02:28,901:DEBUG:UDP target IP: 172.16.1.3  UDP target port: 7780
2019-05-25 12:02:28,908:DEBUG:Sending: "C38,3003,0.0"
```

The Switch is turned turn with the following control code C38,3003,1

You may find being able to acknowledge the warning by hitting the UC Master Warning useful- C24,3001,1

The Yaw Trim potentiometer on the SAS pane is manipulated in a similar way

Control Code	Action
C38,3013,-1	Fully Counter-Clockwise
C38,3013,0	Centered
C38,3013,1	Fully Clockwise
C38,3013,-0.6666	Two-thirds back from Center

Probably the single most challenging thing is trying to work out what button maps to what command string. Fortunately the DCS-BIOS team have documented it. The commands are found in the lib directory (dcs-bios-xx→ Scripts→DCS-BIOS→lib. For the A10C the file is A10C.lua. Search towards the tail end of the file to find the commands.

Aircraft such as the A10C largely focus on mapping switches in panels, the HOTAS commands are present in clickabledata.lua (and DCS-BOIS). They are, however found in command\_defs.lua, which is also in the Mods → aircraft→ A-10C→Cockpit→Scripts folder. As an example here's the values for speedbrake from the A-10C command\_defs

```

Plane_HOTAS_SpeedBrakeSwitchForward = 577,
Plane_HOTAS_SpeedBrakeSwitchAft = 578,
Plane_HOTAS_SpeedBrakeSwitchCenter = 579,
```

The values in commanddefs are not able to be triggered using the same lua mechanism as the ones in clickabledata. With clickabledata we are setting something on a device (panel), switch, switch value. The `performClickableAction` method is performed on the device.

```

lCommandArgs = StrSplit(string.sub(lInput,2),",")
lDevice = GetDevice(lCommandArgs[1])
lLastValue = tostring(lDevice)
if type(lDevice) == "table" then
    lDevice:performClickableAction(lCommandArgs[2],lCommandArgs[3])
end
```

To raise(or lower – need to verify) the landing gear handle the following command is included in the json file, "API\_Open": "C39,3001,0.0". The leading C indicates the perform clickable action will be used.

From <https://github.com/dcs-bios/module-a-10c/blob/master/A-10C.lua>

```

defineElectricallyHeldSwitch("SASP_YAW_SAS_L", 38, 3003, 3004, 185, "SAS Panel", "Yaw SAS Left OFF - ENGAGE")
defineElectricallyHeldSwitch("SASP_YAW_SAS_R", 38, 3005, 3006, 186, "SAS Panel", "Yaw SAS Right OFF - ENGAGE")
defineElectricallyHeldSwitch("SASP_PITCH_SAS_L", 38, 3007, 3008, 187, "SAS Panel", "Pitch SAS Left OFF - ENGAGE")
defineElectricallyHeldSwitch("SASP_PITCH_SAS_R", 38, 3009, 3010, 188, "SAS Panel", "Pitch SAS Right OFF - ENGAGE")
defineTumb("SASP_MONITOR_TEST", 38, 3011, 189, 1, {-1, 1}, nil, false, "SAS Panel", "Monitor Test Left/Right")
definePushButton("SASP_TO_TRIM", 38, 3012, 190, "SAS Panel", "T/O Trim Button")
definePotentiometer("SASP_YAW_TRIM", 38, 3013, 192, {-1, 1}, "SAS Panel", "Yaw Trim")
```

Troubleshooting adjust VHF volume controller

C58,3006,0	Min
C58,3006,0.5	Mid
C58,3006,0.75	Three Quarters
C58,3006,1	Max
VHF Mute	C58,3005,1
VHF Unmute	C58,3005,0

Whereas with `command_defs` we are setting something which is global, the `LoSetCommand` method is used with just a single parameter

```

lCommandArgs = StrSplit(string.sub(lInput,2),",")
LoSetCommand(lCommandArgs[1])
```

To close the speedbrake the following command is configured in the json file. "API\_Open": "P577". The leading P indicates that LoSetCommand is to be used.

### The relevant piece of export.lua

```
lCommand = string.sub(lInput,1,1)

if lCommand == "R" then
    ResetChangeValues()
end

if (lCommand == "C") then

    -- Using Panel Specific commands from clickabledata.lua
    lCommandArgs = StrSplit(string.sub(lInput,2),",")
    lDevice = GetDevice(lCommandArgs[1])
    lLastValue = tostring(lDevice)
    if type(lDevice) == "table" then
        lDevice:performClickableAction(lCommandArgs[2],lCommandArgs[3])
    end
end

- If there is not a specific panel to use (eg HOTAS then use commands from
- command_defs.lua)

if (lCommand == "P") then
    -- data is PXXX where XXX is command integer from command_defs
    lCommandArgs = StrSplit(string.sub(lInput,2),",")
    LoSetCommand(lCommandArgs[1])
End
```

Analog inputs are found in a similar way – additional contributors have provided new aircraft bindings.

[https://github.com/dcs-bios/module-f-16c-50/blob/master/F-16C\\_50.lua](https://github.com/dcs-bios/module-f-16c-50/blob/master/F-16C_50.lua)

### So for the F16

```
--External Lights
defineTumb("ANTI_COLL_LIGHT_KNB", 11, 3001, 531, 0.1, {0, 0.7}, nil, true, "External Lights", "ANTI-COLL Knob,
OFF/1/2/3/4/A/B/C")
defineToggleSwitch("POS_FLASH_LIGHT_SW", 11, 3002, 532, "External Lights", "FLASH STEADY Light Switch,
FLASH/STEADY")
define3PosTumb("POS_WING_TAIL_LIGHT_SW", 11, 3003, 533, "External Lights", "WING/TAIL Light Switch,
BRT/OFF/DIM")
define3PosTumb("POS_FUSELAGE_LIGHT_SW", 11, 3004, 534, "External Lights", "FUSELAGE Light Switch, BRT/OFF/DIM")
definePotentiometer("FORM_LIGHT_KNB", 11, 3005, 535, nil, "External Lights", "FORM Light Knob")
defineTumb("MASTER_LIGHT_SW", 11, 3006, 536, 0.1, {0, 0.4}, nil, true, "External Lights", "MASTER Light Switch,
OFF/ALL/A-C/FORM/NORM")
definePotentiometer("AIR_REFUEL_LIGHT_KNB", 11, 3007, 537, nil, "External Lights", "AERIAL REFUELING Light
Knob")
define3PosTumb("LAND_TAXI_LIGHT_SW", 11, 3008, 360, "External Lights", "LANDING TAXI LIGHTS Switch, LANDING/OFF/
TAXI")
```

To communicate with DCS – export.lua needs to be modified. There is a default export.lua

in the program folders – but you should place you export.lua here:

\$HOME\Saved Games\DCS\Scripts\Export.lua

A working (at least with DCS 2.5) export.lua resides in the git repository ‘Python HW Link → DCS Export → Export.lua’. Copy that to your saved games directory (eg local user here is bnepe).

C:\Users\bnepe\Saved Games\DCS.openbeta\Scripts\Export.lua

The LuaExportStart is called before the mission starts (before the briefing I think). This set’s the UDP ports up and could be used to issue the CQ request to synchronise the virtual cockpit with the physical cockpit. The CQ request should be send to the Raspberry Pi who has knowledge of all physical input modules – which it can then individually forward to each module.

For the purposes of highlighting the most import pieces the comments have largely been removed.

It initially creates a log file and inserts version information (PTExport) in

```
C:\Users\bnepe\Saved Games\DCS.openbeta\logs'

ProductName: DCS
FileVersion: 2.5.4.30038
ProductVersion: 2.5.4.30038
```

It then opens a UDP listening port on 49000 – this is used to receive commands either from the Pi (preferred) but also could be direct. The name of the socket variable is CDU as this code was originally used for the CDU interface by Levinsky. There is an absolute minimum wait associated with the socket as we have not need to hang around waiting for command to arrive – there is always the game frame.

```
function LuaExportStart()

    default_output_file = io.open(lfs.writedir().."/Logs/PTExport.log", "w")

    package.path  = package.path..";"..lfs.currentdir().."/LuaSocket/?..lua"
    package.cpath = package.cpath..";"..lfs.currentdir().."/LuaSocket/?..dll"
    socket = require("socket")

    local version = LoGetVersionInfo()
    if version and default_output_file then

        default_output_file:write("ProductName:..version.ProductName..'\n')
        default_output_file:write(string.format("FileVersion: %d.%d.%d.%d\n",
            version.FileVersion[1],
            version.FileVersion[2],
            version.FileVersion[3],
            version.FileVersion[4]))

        default_output_file:write(string.format("ProductVersion: %d.%d.%d.%d\n",
            version.ProductVersion[1],
            version.ProductVersion[2],
            version.ProductVersion[3],
            version.ProductVersion[4]))
```

```

        version.ProductVersion[1],
        version.ProductVersion[2],
        version.ProductVersion[3], -- head revision
        version.ProductVersion[4])) -- build number
    end

    -- Using Same Receive port as Xplane for Python HW InterfaceAlias
    -- The CDU variable is a hang over from the work of Levinsky with A10 CDU
    cdu_receive_port = 49000
    c_socket = require("socket")
    cdu_socket = c_socket.udp()
    cdu_socket:setsockname("*",49000)
    cdu_socket:setoption('broadcast',true)
    cdu_socket:settimeout(.001) -- set timeout for reading socket

package.path  = package.path..";..\LuaSocket\\?.lua"
package.cpath = package.cpath..";..\LuaSocket\\?.dll"

socket = require("socket")

gps_export_port = 13136
gps_export_host = "172.16.1.2"
gps_export_socket = require("socket")
gps_export_con = socket.try(gps_export_socket.udp())
gps_export_socket.try(gps_export_con:settimeout(.001))
gps_export_socket.try(gps_export_con:setpeername(gps_export_host,gps_export_port))

end

```

The **LuaExportStop** is used to close files, sockets, and also can be used to set gauges and lamps to a default position. This can be useful for gauges than are unable to auto sense a zero position such as the SimMeters Altimeter.

```

function LuaExportStop()
    if default_output_file then
        default_output_file:write("Closing: ..'\n')
        default_output_file:close()
        default_output_file = nil
    end
end

```

As the name suggests the **LuaExportBeforeNextFrame** is fired before each frame. Here function pulls a packet from the listen socket, and then looks to see if the payload is either a Reset (which current is not in us), a Command (C) , or a P which I think is used for things like HOTAS -(need to validate)

```

function LuaExportBeforeNextFrame()
    local lInput = cdu_socket:receive()
    local lCommand, lCommandArgs, lDevice, lArgument, lLastValue

    if lInput then
        lCommand = string.sub(lInput,1,1)
        if lCommand == "R" then
            ResetChangeValues()
        end

        if (lCommand == "C") then

```

```

-- Using Panel Specific commands from clickabledata.lua
lCommandArgs = StrSplit(string.sub(lInput,2),",")
lDevice = GetDevice(lCommandArgs[1])
lLastValue = tostring(lDevice)
if type(lDevice) == "table" then
    lDevice:performClickableAction(lCommandArgs[2],lCommandArgs[3])
end
end

if (lCommand == "P") then
    lCommandArgs = StrSplit(string.sub(lInput,2),",")
    LoSetCommand(lCommandArgs[1])
end
end
end

```

The LuaExportActivityNextEvent is used to export data to the physical cockpit. This is used in preference to the ExportBeforeFrame as we really want a steady state of data towards the instruments, and typically this doesn't need to fire more than ten times a second.

```

function LuaExportActivityNextEvent(t)

local tNext = t
local gps_export_flightData = {}

-- For this document have removed commands that are associated
-- with why certain conversations are made

table.insert(gps_export_flightData,"latitude"..LoGetSelfData().LatLongAlt.Lat)
table.insert(gps_export_flightData,"longitude"..LoGetSelfData().LatLongAlt.Long)
table.insert(gps_export_flightData,"altitude"..LoGetAltitudeAboveSeaLevel())

table.insert(gps_export_flightData,"airspeed"..LoGetTrueAirSpeed() * 1.94)

table.insert(gps_export_flightData,"magheading"..LoGetSelfData().Heading * 57.3)

gps_export_packet = gps_export_flightData[1] .. "," ..
    gps_export_flightData[2] .. "," .. gps_export_flightData[3] .. ","
gps_export_packet = gps_export_packet .. gps_export_flightData[4] .. "," ..
    gps_export_flightData[5]

gps_export_socket.try(gps_export_con:send(gps_export_packet))

tNext = tNext + 0.5 --repeat every 1/2 second

return tNext
end

```

The sample export.lua has a function to split a received string into a number of entries, allowing multiple commands to be passed to DCS in a single UDP packet

```

function StrSplit(str, delim, maxNb)

-- Eliminate bad cases...
if string.find(str, delim) == nil then
    return { str }
end

if maxNb == nil or maxNb < 1 then

```

```
    maxNb = 0      -- No limit
end

local result = {}
local pat = "(.-)" .. delim .. "()"
local nb = 0
local lastPos

for part, pos in string.gfind(str, pat) do
    nb = nb + 1
    result[nb] = part
    lastPos = pos
    if nb == maxNb then break end
end

-- Handle the last field
if nb ~= maxNb then
    result[nb + 1] = string.sub(str, lastPos)
end

return result
end
```

# X-Plane

Originally tried to use the API interface that allowed reading and writing to variables, found the reading worked very well for subscribing to a data stream from the Simulator.

UDP Port	Use
49000	X-Plane receives commands
49001	X-Plane sends UDP payloads
49002	X-Plane to iPad

Simply select the values of interest, (eg Speeds, Lat,Long, Altitude) in the Telemetry screen.

A warning will be presented that the Network Data Output hasn't been configured. Enter IP Address and target port (as of 20181229 using common port 13136 - but this is likely to change)

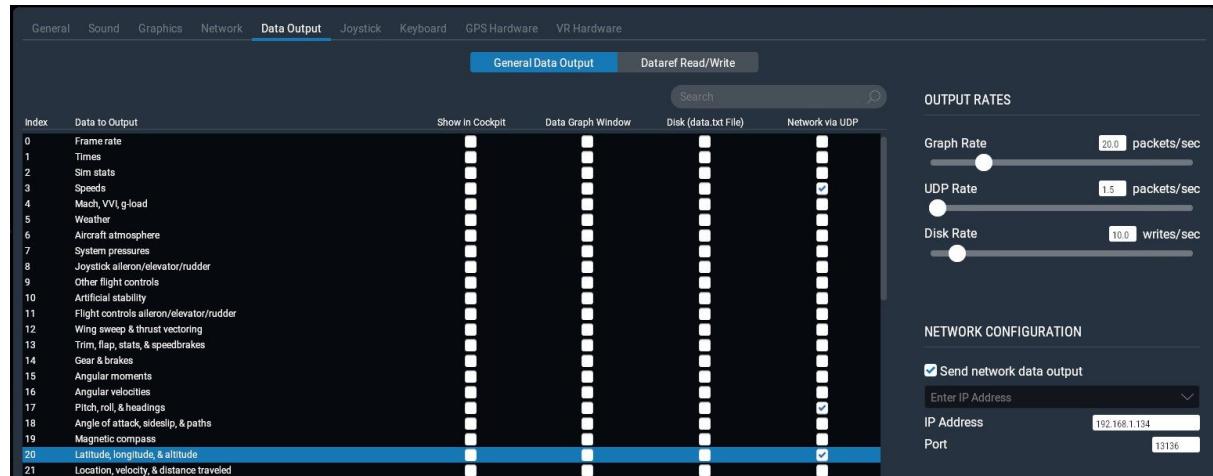


Figure 2: X-Plane - Selecting Indexes to export

The following settings are used for driving the GPS

3	Speeds	<input type="checkbox"/>
4	Macn, vvi, g-load	<input checked="" type="checkbox"/>
5	Weather	<input type="checkbox"/>
6	Aircraft atmosphere	<input type="checkbox"/>
7	System pressures	<input type="checkbox"/>
8	Joystick aileron/elevator/rudder	<input type="checkbox"/>
9	Other flight controls	<input type="checkbox"/>
10	Artificial stability	<input type="checkbox"/>
11	Flight controls aileron/elevator/rudder	<input type="checkbox"/>
12	Wing sweep & thrust vectoring	<input type="checkbox"/>
13	Trim, flap, stats, & speedbrakes	<input type="checkbox"/>
14	Gear & brakes	<input type="checkbox"/>
15	Angular moments	<input type="checkbox"/>
16	Angular velocities	<input type="checkbox"/>
17	Pitch, roll, & headings	<input checked="" type="checkbox"/>
18	Angle of attack, sideslip, & paths	<input checked="" type="checkbox"/>
19	Magnetic compass	<input checked="" type="checkbox"/>
20	Latitude, longitude, & altitude	<input checked="" type="checkbox"/>

Figure 3: X-Plane GPS Indexes selected

The mapping of values is described here <https://www.x-plane.com/kb/data-set-output-table/>

Selecting the three indexes results in there being three record sets sent per update (in a single packet). A very useful site that explains how this hangs together is <http://www.nuclearprojects.com/xplane/receivedata.shtml>.

Whilst it specifically discussed coding in c#, the packet format on the wire is the common.

All data is sent as bytes

- There are 41 bytes per sentence
  - The first 5 bytes are the message header, or "prologue"
    - First 4 of the 5 prologue bytes are the message type, like "DATA"
    - Fifth byte of prologue is an "internal-use" byte
  - The next 36 bytes are the message
    - First 4 bytes of message indicates the index number of a data element, as shown in the Data Output screen in X-Plane
    - Last 32 bytes is the data, up to 8 single-precision floating point numbers (4 bytes per floating point number)

Using that API that was used for reading values don't work so nicely in the opposite

direction. This was resolved relatively simply, over googling around, found the needed information was sitting with the X-Plane installation itself.

```
D:\Xplane11\X-Plane 11\Instructions\X-Plane SPECS from Austin\Exchanging  
Data with X-Plane.rtf
```

### Some important Tips from this guide

NOTE: X-Plane always receives on port 49000.

NOTE: Any strings that you send should be null-terminated!

To send a UDP message to X-Plane:

- the 4-letter label
- a byte of value '0'
- the message data you want to send

RUN A COMMAND: CMND

DATA INPUT STRUCTURE is a string

```
CMND0+sim/flight_controls/flaps_up
```

The data part of this message is simply the command that you want X-Plane to initiate!

To see available X-Plane commands, run X-Plane and go to the Settings menu, Keyboard tab or the Joystick Tab.

The commands are the group name you see in the centre of the screen, PLUS command string in the right side, all run together.

Commands can be seen by hovering over the desired action

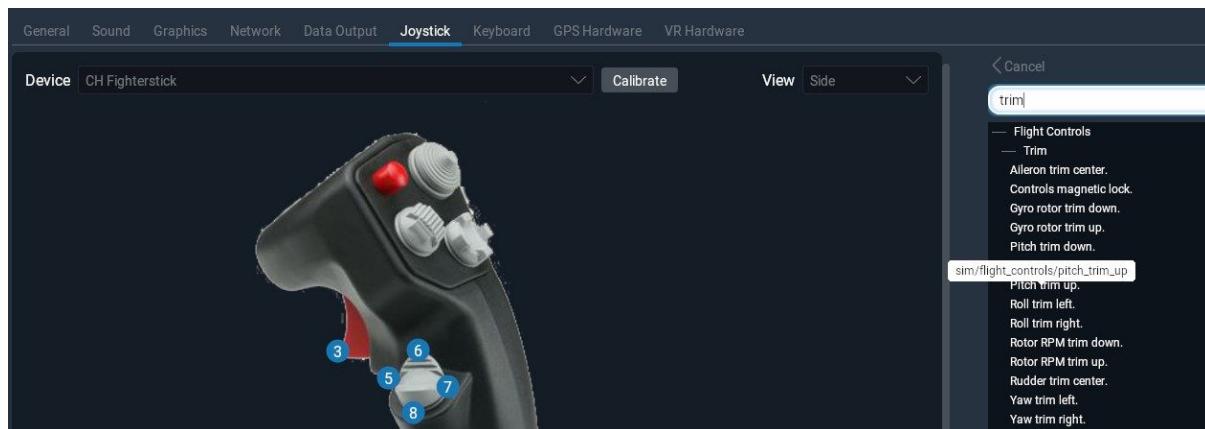


Figure 4: X-Plane Joystick showing commands

Or via the keyboard

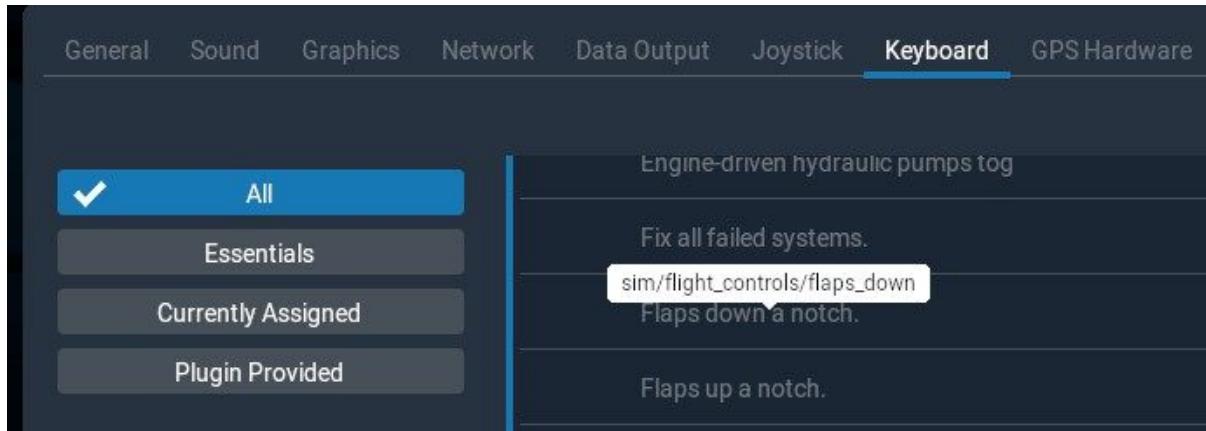


Figure 5: X-Plane Keyboard showing commands

To set values in X-Plane you can explicitly write to a DREF, this blog provides a very useful guide - <https://blog.shahada.abubakar.net/post/linux-udp-network-client-library-for-x-plane-10-and-11/>

**BECN** - Beacon broadcast. There is no equivalent feature for this in ExtPlane. Every running instance of X-Plane will broadcast its IP address and command port number on your LAN in a BECN message, every second. With this, it is possible for devices on your LAN to "auto-discover" the IP and port number of the X-Plane server. It is also possible to auto-detect if there are multiple X-Plane servers running on your LAN, and offer the user a choice of which one they wish the device to connect to. It's a lot better than having the user manually configure the device with the IP and port number of the server, and helps a lot when you have more than one X-Plane PC. Note that this is a multicast datagram, so you will need to "subscribe" to the multicast address 239.255.1.1 port 49707.

**RREF** - Request subscription to a DataRef. The client sends this message to X-Plane's command port, together with a frequency (in Hz) and an identifying number. X-Plane will then automatically send the client a response RREF message at the frequency specified containing the current value of that DataRef. To unsubscribe, you send the RREF again with a frequency of 0.

**DREF** - Set the value of a DataRef. You send a DREF together with the value you want, and the name of the dataref.

**CMND** - Gets X-Plane to execute a command. You send a CMND together with the name of the Command you want invoked, and X-Plane executes it.

## Choice of platform for the hardware interface.

Whilst the Raspberry Pi offers the nicest development and troubleshooting environment, it lacks the high pincount found on an Arduino. The Arduino will require an Ethernet shield (and not all Ethernet shields are created equal, have run into issues with an IOT shield that had a poorly cooled chip which caused lock ups and see Attack of the Clones towards the end of the document for other some tips).

As the Arduino codes runs without an intermediate operating system, it offers the highest performance for IO related tasks such as driving stepper motors.

Have ran into issues with lockups in the 737 overhead display, possibly due to incorrectly

terminated strings or invalid characters included in string. Strongly bounds checking will be performed on the Pi before strings are send to Arduino displays.

As we are not bound by the 128 input limit associated with either windows DirectX or the 32 input limit associated with FSUIPC, OverPro's Arduino Joystick interface can be used with 256 inputs reducing the number of controllers needed for the pit. Hopefully a single controller for Port, Starboard and Forward zones can be used. OverPro's code will be modified removing the USB interface, instead storing switch state and reporting deltas after completing a full scan of 256 inputs. Unsure if rotaries will be supported, no logical reason why not.

## Program tasks

The workload is divided into two programs – one dealing with inputs, the other with outputs. Both are fundamentally loops which briefly block awaiting receipt of a UDP packet either from the simulator/shim or from the input devices.

Command line parameters are used to get a debugging level as well as enable a configuration mode.

Configurations will be held in two separate files, the format of these files is yet to be determined, but JSON is mostly likely. The configuration files include:

1. IP Address code listens on (optional - if not explicitly configured 127.0.0.1)
2. Port Code listens on (optional – if not explicitly configured 7784 for input module and 7785 for output module. (still considering whether to leave export.lua sending direct to existing Arduino units)
3. Mapping of input to aircraft commands and aircraft outputs to physical displays/gauges.

## Protocols

Where possible output values will be carried directly mapping to value to be displayed, and may be integer, float, or string. Indicators/Solenoids will be represented as a 0 or 1.

Data will be carried as AV pairs A1:V2, A2:V2. The data packet will be preceded with a D.

Operational tasks such as shutdown, reboot, refresh switch state will be preceded with a C (command), and a single operational task. e.g. C9999.

### Operational Task

Task Id	Task	

CQ	Send all switch states	

Ikarus uses the following format

```
# If Sending Commands to DCS with Ikarus installed
# The values to be sent can be found
# C:\Program Files\Eagle Dynamics\DCS World\Mods\aircraft\Uh-1H\Input\UH-1H\joystick
# Structure is
# C - Command
# 15 - Cockpit Device Id
# 3003 - Unknown but seen in multiple places
# Switch Position
# Send to Port UDP_PORT = 26027
# MESSAGE = "C15,3003,-1" - Turns Test Switch on - All warning lights
# MESSAGE = "C15,3003,0" - Turns Test Switch to centre
# MESSAGE = "C15,3003,1" - Turns Test Switch to Reset - clears caution on front panel
# Bright/Dim 15,3004
```

## Learning's from Different Modules

## Determining which Simulator is Operational

Ideally the number of changes made to 'downstream' modules should be minimised. As X-Plane packets always originate from 49001, this can help distinguish which Sim is running. The following approach was used in the GPS code.

```
# Need to decode the payload to convert from bytes object to a string
# Don't do this is the packet is from XPlane Sourced from 49001
if (Source_Port != 49001):
    # SimConnect P3d
    SendingSim = 'SimConnect'
    ReceivedPacket = data.decode()
    ReceivedPacket = str(ReceivedPacket)
else:
    SendingSim = 'XPlane'
    ReceivedPacket = data
```

# Raspberry Pi

The USB.Core library provides a low level interface for the Raspberry Pi. It is able to scan the USB bus and report back what is attached.

The code scans the USB bus looking for a specific device, and then detaches the device from the Kernel driver. Once this is complete it then runs a set configuration, and then asks for a data block from the target device on a frequent interval.

A quick check is made to see if the value returned is different from the last interval, if it is then the list holding switch positions is refreshed

## Maintaining the Pi

```
pi@GenFrontPi:~ $ sudo apt-get upgrade
Reading package lists... Done
Building dependency tree
Reading state information... Done
Calculating upgrade... Done
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
pi@GenFrontPi:~ $ sudo apt-get dist-upgrade
Reading package lists... Done
Building dependency tree
Reading state information... Done
Calculating upgrade... Done
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
pi@GenFrontPi:~ $
```

The Pi now supports the Arduino IDE. To install:

```
sudo apt-get update && sudo apt-get upgrade
sudo apt-get install arduino
```

Unlike the Windows or Mac IDE, the Pi IDE doesn't appear to support the online libraries. Needed to download the Led Control Library, unzip it, and then upload from source. It didn't like special characters in the library name. As multiple Arduinos are connected to the hub, its important to be care about which device you are programming, generally easiest to just the non-target Arduinos from the USB hub.

## Initiating Scripts on Pi Nodes

Instead of trying to start python directly from crontab, use shell script (usually my\_server) to start things

To get the script to autostart

```
sudo crontab -e
```

And add line

```
@reboot sh /home/pi/Documents/Flightsim/Huey\ Caution\ Panel/my_server 2>&1
```

Which results in the crontab file looking like

```
# Edit this file to introduce tasks to be run by cron.  
..  
..  
#  
@reboot sh /home/pi/Documents/Flightsim/Huey\ Caution\ Panel/my_server 2>&1
```

Originally a separate shutdown script was operated, need to work out why it was commented out, possibly as the script below only ever reaches remoteshut.py after the receiver code exits, which is never...

```
### BEGIN INIT INFO  
# Provides: my_server  
# Required-Start: $remote_fs $syslog $network  
# Required-Stop: $remote_fs $syslog $network  
# Default-Start: 2 3 4 5  
# Default-Stop: 0 1 6  
# Short-Description: Simple Web Server  
# Description: Simple Web Server  
### END INIT INFO  
  
#! /bin/sh  
# /etc/init.d/my_server  
  
export HOME  
echo "Starting My Server"  
cd /home/pi/Documents/Flightsim/Huey\ Caution\ Panel  
sudo /usr/bin/python receiver_004.py 2>&1 &  
#sudo /usr/bin/python remoteshut.py 2>&1 &  
exit 0
```

## Receive UDP Port utilisation

A number of these values are referenced from soic\_conv\_ExportStart.lua from DCS

Port	IP Address	Description
	127.0.0.1	New DCS Emulator
7777	127.0.0.1	SOIC on Primary Sim PC <sup>6</sup>

<sup>6</sup> The SOIC port points to another Shim which maintains a TCP connection to the SOIC processes, converting the UDP payload into a TCP stream. Currently error handling does not address a restart of SOIC processes.

Port	IP Address	Description
7784		Input Codes listens on
7788	127.0.0.1	Arduino Sender Emulator. Sends to Pri_Node_Input
7788	127.0.0.1	Arduino Sensor. Sends to Pri_Node_Input
7789	127.0.0.1	GUI_Sender. Sends packets to Pri_Node_Input
7790	PC Running Sim	Keystroke_Sender
7791	127.0.0.1	Lamp_Output_Emulator
7792	127.0.0.1	Display_Output_Emulator
7793	127.0.0.1	XPlane_Decode
7794	127.0.0.1	Radio_Control
7795	127.0.0.1	USB_Reader
13135	192.168.1.105	Fuel Hands on A10
13135	192.168.1.106	Fuel Display (OLED on A10)
13135	192.168.1.107	Compass and Clock Analog hands
13135	192.168.1.108	Clock Digits
13135	192.168.1.109	General Stepper
13135	172.16.1.21	General_Sim_7219
26027	127.0.0.1	Pri_Node_Input
26028	127.0.0.1	Pri_Node_Output
27000	127.0.0.1	UDP_Reflector
49000	127.0.0.1	X-Plane UDP listener

#### IP Addressing – Backend 172.16.1.X

Backend		
172.16.1.2	Primary Pi	
Input Devices		
172.16.1.10	Left Arduino – Device id 01	0x00,0xDD,0x3E,0xCA,0x35,0x02
172.16.1.11	Front Arduino – Device id 03	0x00,0xDD,0x3E,0xCA,0x35,0x03
172.16.1.12	Right Arduino – Device id 03	0x00,0xDD,0x3E,0xCA,0x35,0x04
172.16.1.13-17		
Output		

172.16.1.20	Left Led Arduino	0x03,0xC7,0x3E,0xCA,0x35,0x02
172.16.1.21	Front Led Arduino	0x03,0xC7,0x3E,0xCA,0x35,0x03
172.16.1.22	Right Led Arduino	0x03,0xC7,0x3E,0xCA,0x35,0x04
172.16.1.23-27		

## Adding RS232 Interface

Ref - <https://www.instructables.com/id/Read-and-write-from-serial-port-with-Raspberry-Pi/>

Using Max 3232 based interface which supports 3.3 to 5V

Pin Used (remembering pin numbers does include both sides of header – so these pins are adjacent.

- 4 (5V),
- 6 (GND)
- 8 (TX),
- 10 (RX)

Can use sudo raspi-config → Interfacing Options → Serial to enable/disable console through serial interface. In the disti used in Dec 2018 it is disabled by default – enabled to validate the TX led flashes during the reload process.

## Notes from the Huey readme

Git Commands

Clone Repo

git clone <https://github.com/bnepethomas/bne-arduino-flight-simulator-interfaces.git>

```
git config --global user.email "your email"
git config user.name "your name"
```

Watch for untracked files

```
$ git add .
$ git commit .
$ git push
```

## System Flows

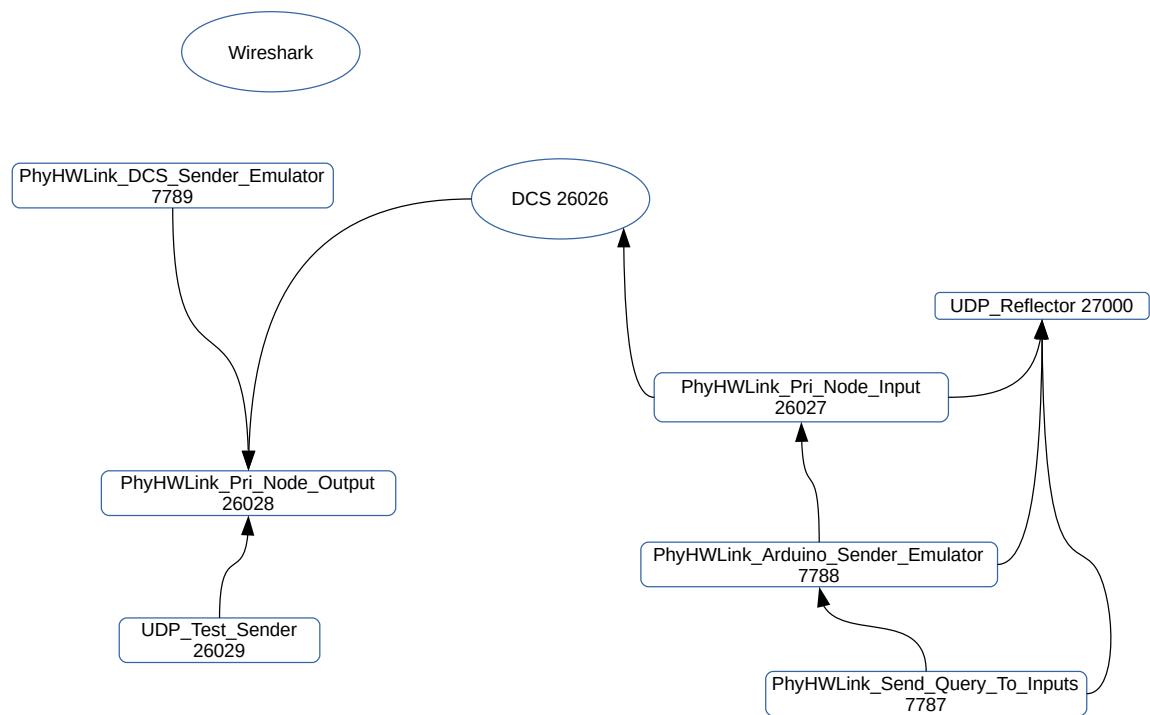


Figure 6: System Flows

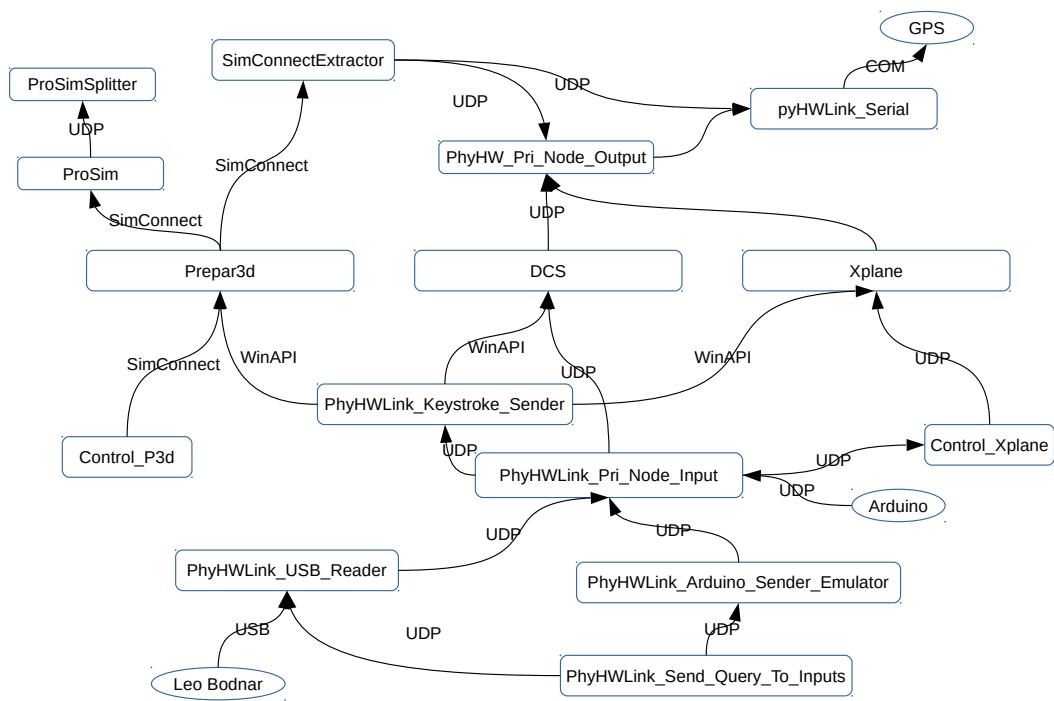


Figure 7: Detailed System Flows

## Code Modules

### General Sim 7219

This Arduino sketch provides an interface to Servo, General I/O and Max7219 powered Leds.

It is intentionally simple, with 'fancy' work performed by PyHWLink\_Pri\_Node\_Output.

Data Packet – AV pairs preceded by a 'D'

D,1:0,2:1,3:1,4:1,5:0,6:0,7:0,8:0,9:1,10:1

All outputs accept only integer values.

Input Data Ranges	Outputs	Pins	Total Outputs
1-64 (65-128 Rsvd)	Led outputs driving		
150-	Servos	21-39	18
200-208	Digital I/O	40 - 48	9

Listens to	Port
p3d_PyHWLink_Pri_Node_Output	13135

### Hornet Altimeter

The Altimeter project is broken into two pieces – the code to drive the needle, and the code to drive the OLED. Originally the code was all run on an Arduino, but largely due to my lack

of skills in dealing with an environment that needs to be interrupt driven, the workload is split between an Arduino for driving the motor, and a Raspberry Pi for the OLED code.

## Altimeter OLED

The Altimeter OLED displays the Altitude (ten thousands, thousands, hundreds) and the pressure setting (QNH). The last two digits will either not be displayed

The Altimeter OLED code is based on sample code from Adafruit. Hardware interfacing details are found later in this document.

'Rolling' characters are used, which do increase the workload but considerably improve the realism of the altimeter. The minor digits (ones and tens) are now updated, they are left as '0's which may either be engraved in the faceplate or just simply rendered as '0's. The major digits (ten thousands, thousands, and hundreds) are updated and do roll.

There are two special use cases, the left hand hash, which sits in the ten thousands column, and the value at which digits roll. The digits should hold a centered position until they are close to changing value, and only then do they roll. The roll is done within the last 100 feet. As the hash is different height to characters it has its own position value (hatch\_top). It is located on the ten thousands column.

Drawing the hash. Whilst an option was considered to use a predrawn graphic for the hatch, it ultimately was not a large coding effort to draw the hatch. This is done by drawing a white box and then drawing a series of black lines

PIL is used for the graphical elements.

Characters are drawn using column spacing, with a spacing currently of 23 pixels. Pixels are used to locate elements on the canvas. On the X co-ordinates numbers increase from left to right. Row positions are a little more interesting, with negative values used for the top most rows.

## PyHWLink\_Arduino\_Sender\_Emulator

Emulates a Arduino running OverPro's Joystick interface with an Ethernet shield.

As we aren't exposed to any Joystick button limits, there are 256 buttons supported

Instead of using the stub to provide a Joystick via USB, delta are sent to a process running on a Raspberry Pi which maps these deltas to commands for the Sim

The Arduino code is independent of the aircraft and simulator that is running, which simplifies its operation.

There is a receive operation where the Arduino will report the state of all 256 inputs. This will be spread across several packets with a 300mS delay between # the packets. The trigger packet is a simple 'CQ'

It is configurable, with the Input\_Module\_Number, which determines which input module it is emulating, the max packet size (basically to throttle the number of entries in a single packet), which hopefully throttles the number of entries concurrently hitting the downstream modules.

The likelihood of change determines the rate of change of 'input' values in a given cycle.

Sends to	Port
PyHWLink_Pri_Node_Input	26027
UDP_Reflect	27000

Listens to	Port
PyHWLink_Send_Query_To_Inputs	26027

The Packet format is a D followed by Module\_Number, Switch\_No, Current\_Switch\_State

The packet payload is not fixed length -

D00:003:0,00:005:0,00:006:1,00:007:1,00:009:1

## **PyHWLink\_Display\_Output\_Emulator**

This module receives UDP packets containing desire output state for displays such as OLED or 7 Segment Led..

Listens to	Port
PyHWLink_Pri_Node_Output	7792

b'D,1:0,2:1,3:1,4:1,5:0,6:0,7:0,8:0,9:1,10:1,11:

## PyHWLink\_GUI\_Sender

As the name suggests this module was developed to provide a GUI to send simulated button/switch transitions to Pri\_Node\_Input. It uses the tkinter library that provides platform support for MacOS, Windows and the Raspberry Pi. Only MacOs required an install of the tkinter environment.

There are two types of button actions provided:

- 1: Classic Toggle Switch Action where switch position is held, and
- 2: A Push button, where a simulated Push and Release action is transmitted with a relatively short delay. Its up to Pri\_Node\_Input to determine if it will act only on the Press or on both Press and Release (as may be needed if Key Strokes are being sent)

The same packet format is used as the Arduino\_Sender\_Emulator

b'D01:005:1'

The user can select which module is being emulated (1-5)

Sends to	Port
PyHWLink_Pri_Node_Input	26027
UDP_Reflect	27000

The GUI is manually laid out.

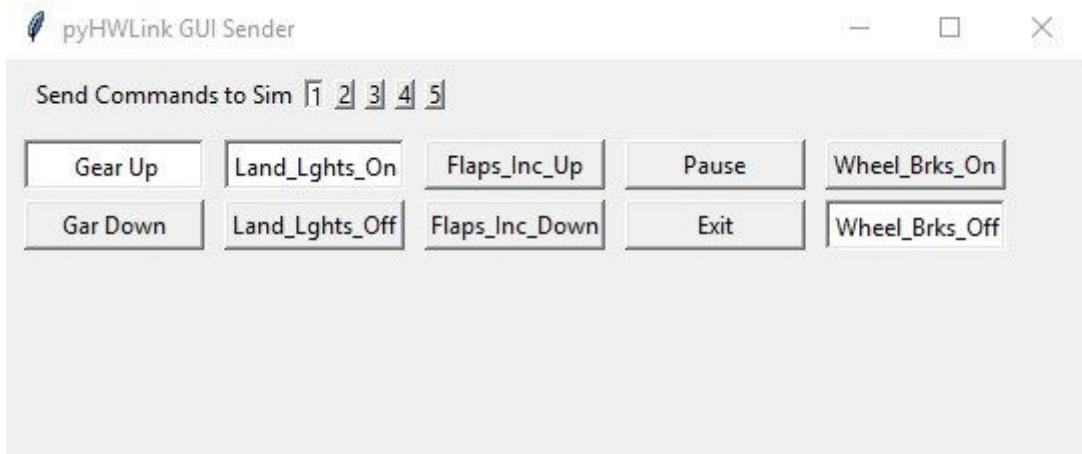


Figure 8: PhyHWLink\_GUI\_Sender GUI

Switch No	Role	Pressed	Released	Temp or Held
1	Gear	Gear Down	Gear Up	Held
2	Landing Lights	Lights On	Lights off	Held
3	Flaps Increment Up	Flaps Inc Up		Temp
4	Pause Sim	Pause Sim		Temp
5	Exit Sim	Exit Sim		Temp
6	Wheel Brakes	Brakes On	Brakes Off	
7	Flaps Increment Down	Flaps Inc Down		
8				
9				
10				
11				
12				
13				
14				
15				
16				
17				

Switch No	Role	Pressed	Released	Temp or Held
18				
19				
20				
21				
22				
23				
24				
25				
26				
27				
28				
29				
30				
31				
32				
33				
34				
35				

## PyHWLink\_Lamp\_Output\_Emulator

This module receives UDP packets containing desire output state for lamps. It may be extended to receive string information for airspeed, radios etc, but initially just aimed at displaying output state.

Listens to	Port
------------	------

PyHWLink_Pri_Node_Output	7791

This only has two fields as it emulates a single end point. The same packet format is used as the Display\_Output\_Emulator

D,1:0,2:1,3:1,4:1,5:0,6:0,7:0,8:0,9:1,10:1

## PyHWLink\_Keystroke\_Sender

This module receives characters to be send to whatever application has focus, which should ultimately be the flight sim. It uses a low level API to increase the likelihood interoperability with DirectX based games.

Did struggle with the code only appearing to work on simple Applications such as notepad and also X-Plane. Nothing received in DCS or P3d. Then discovered windows appears to have added some security which appears to stop the API working with DirectX games.

<https://www.tenforums.com/software-apps/49635-sendkeys-not-working-windows-10-a-3.html>

SendKeys are locked in Win 10 and 8.1 by the UAC. If you need to use Sendkeys you may want to try Disabling it in the Registry :

HKLM>Software>Microsoft>Windows>CurrentVersion>Policies>System>EnableLUA=0.

Also needed to use DirectX mappings which is different to classic keycodes. Have held original keycodes just in case the directX doesn't work with other games.  
After restarting due to the change, things worked on all applications I tested.

Found best way to test if characters are being send was to go into DCS, → Options -> Controls Keyboard. Select an action and then play characters into that.

Sends to	Port
Whatever Application currently has focus	Windows API for local machine

Listens to	Port

PyHWLink_Pri_Node_Inputs (and derivatives)	7790

## PyHWLink\_Pri\_Node\_Input

This module is the input workhorse of the simulator infrastructure. It receives AV pairs from input devices/process over UDP, translating them into either commands that are sent direct to the Sim, or are send to Control\_Via\_Keyboard, which as the name suggests injects keystrokes into the Simulator using the Windows API.

It has a learning mode which, as the name suggests, listens to events, and enables the user to assign a task to an event. Events are stored in 'input\_assignments.json'.

1. To enable learning change the learning value in 'input\_config.py' to True."

Pri\_Node\_Input is reflector enabled, which means it sends a copy of its outputs to the UDP\_Reflector process for forwarding to Wireshark as needed.

```

Learning Mode: True
Aircraft is: A10C
2019-01-13 15:37:24,377:DEBUG:Checking Command Line parameters
2019-01-13 15:37:24,377:DEBUG:options:{'optionLearning': False}
2019-01-13 15:37:24,377:DEBUG:arguments:[]
Loading Input Assignments from: "input_assignments.json"
Waiting for packet
2019-01-13 15:37:32,984:DEBUG:Message: b'D00:007:1,00:008:1'
2019-01-13 15:37:32,984:DEBUG:Processing UDP String
2019-01-13 15:37:33,266:DEBUG:Message: b'D00:000:0,00:002:1,00:006:1'
2019-01-13 15:37:33,266:DEBUG:Processing UDP String
2019-01-13 15:37:33,562:DEBUG:Message: b'D00:006:0,00:009:1'
2019-01-13 15:37:33,562:DEBUG:Processing UDP String
2019-01-13 15:37:33,844:DEBUG:Message: b'D00:001:0,00:002:0,00:006:1'
2019-01-13 15:37:33,844:DEBUG:Processing UDP String
2019-01-13 15:37:34,140:DEBUG:Message: b'D00:001:1,00:002:1,00:003:1,00:005:1,00:009:0'
2019-01-13 15:37:34,140:DEBUG:Processing UDP String

```

Figure 9: PhyHWLink\_Pri\_Node\_Input User Interface

## **PyHWLink\_Pri\_Node\_Output**

This module is the output workhorse of the simulator infrastructure. It receives AV pairs from Simulator, and then translates these to hardware modules

Sends to	Port
172.16.1.21	13135

Listens to	Port
SimConnect_to_IP	26028

## **pyHWLink\_Radio\_Control**

This module listens to state updates from USB\_Reader. It is currently in a experimental stage with the following goals:

1. Locally Store State (such as current radio frequency) and either directly or incremental update Sim state. Ideally direct updates will occur, which should be possible with P3d, but unsure if DCS allows direct setting of radios, or if button commands will be needed to increment radio channels
2. Enable different roles to be assigned to a single rotary encoder based on whether the encoder is pressed or not
3. Determine how to ‘accelerate’ updates of a rotary encoder. This is largely for roles where precision is required, but a large number of values must be moved through, such as heading and airspeed settings on the autopilot panel.

Sends to	Port

TBA	

Listens to	Port
PyHWLink_USB_Reader	7794

## pyHWLink\_Serial

This module is probably better called pyHWLink\_GPS, initially named to develop a Raspberry Pi serial stack. The single module supports DCS, P3d and X-Plane simultaneously. In part this is due to a common on the wire format used for P3D (as information comes via a SimConnect shim).

X-Plane is a little different over the wire as data is packed using a binary format.

Sends to	Port
Serial Port on Pi	

Listens to	Port
DCS, XPlane, and lightweight SimConnect	13136

Shim	

## DCS Interfacing Code

As DCS is a API/Script Native, there was little extra to do to export information to drive GPS. The same target port and X-Plane was used (13136)

The initialisation code largely is involved with configuring the socket to use.

```
package.path  = package.path.."\\LuaSocket\\?.lua"
package.cpath = package.cpath.."\\LuaSocket\\?.dll"

socket = require("socket")

gps_export_port = 13136
gps_export_host = "192.168.1.135"
gps_export_socket = require("socket")
gps_export_con = socket.try(gps_export_socket.udp())
gps_export_socket.try(gps_export_con:settimeout(.001))
gps_export_socket.try(gps_export_con:setpeername(gps_export_host,gps_export_port))
```

*Text 1: DCS LUA Initialisation - Code Export.lua*

The Event code clears a table and progressively add entries to the table. Once the 5 rows are populated, they are assembled and send out the socket. The event is then rescheduled to fire again in 500mS.

```

function LuaExportActivityNextEvent(t)

local tNext = t
local gps_export_flightData = {}

table.insert(gps_export_flightData,"latitude:"..LoGetSelfData().LatLongAlt.Lat)
-- LATITUDE

table.insert(gps_export_flightData,"longitude:.."..LoGetSelfData().LatLongAlt.Long)
-- LONGITUDE

table.insert(gps_export_flightData,"altitude:.."..LoGetAltitudeAboveSeaLevel())
-- ALTITUDE SEA LEVEL(MTS TO FT)

table.insert(gps_export_flightData,"airspeed:.."..LoGetTrueAirSpeed() * 1.94)
-- TRUE AIRSPEED (M/S TO KNOTS)

table.insert(gps_export_flightData,"magheading:.."..LoGetSelfData().Heading * 57.3)
-- HEADING (RAD TO DEG)

gps_export_packet = gps_export_flightData[1] .. "," ..
gps_export_flightData[2] .. "," .. gps_export_flightData[3] .. ","
gps_export_packet = gps_export_packet .. gps_export_flightData[4] .. "," ..
gps_export_flightData[5]

gps_export_socket.try(gps_export_con:send(gps_export_packet))

tNext = tNext + 0.5 --repeat every 1/2 second
return tNext
end

```

### *Text 2: DCA LUA Event Code - Export.lua*

The packet format uses ',' separators which ':' as a demarcation for attribute name and values.

	6863 170.148242	192.168.1.138	192.168.1.135	UDP	169 54284 → 13136 Len=127
0000	b8 27 eb 00 e2 14 34 e1	2d a0 b4 73 08 00 45 00		'.....4.....s...E..	
0010	00 9a a6 57 00 00 80 11	00 00 c0 a8 01 8a c0 a8		.....W.....	
0020	01 87 d4 0c 33 50 00 86	84 f9 6c 61 74 69 74 75		....3P... latitu	
0030	64 65 3a 34 32 2e 34 31	33 33 36 38 35 32 30 38		de:42.41 33685208	
0040	34 35 2c 6c 6f 6e 67 69	74 75 64 65 3a 34 31 2e		45,longi tude:41.	
0050	37 34 31 33 36 35 34 32	34 35 39 31 2c 61 6c 74		74136542 4591,alt	
0060	69 74 75 64 65 3a 36 35	33 2e 32 35 37 38 37 33		itude:65 3.257873	
0070	35 33 35 31 36 2c 61 69	72 73 70 65 65 64 3a 32		53516,ai rspeed:2	
0080	37 33 2e 35 33 38 30 37	35 38 36 36 37 2c 6d 61		73.53807 58667,ma	
0090	67 68 65 61 64 69 6e 67	3a 32 38 30 2e 35 30 38		gheading :280.508	
00a0	33 39 36 37 38 30 34 39			39678049	

## X-Plane

The name data output is packed binary. Once the needed values are selected for output, X-

Plane exports them in a packed format.

3	Speeds				
4	Macn, vvi, g-load				
5	Weather				
6	Aircraft atmosphere				
7	System pressures				
8	Joystick aileron/elevator/rudder				
9	Other flight controls				
10	Artificial stability				
11	Flight controls aileron/elevator/rudder				
12	Wing sweep & thrust vectoring				
13	Trim, flap, stats, & speedbrakes				
14	Gear & brakes				
15	Angular moments				
16	Angular velocities				
17	Pitch, roll, & headings				
18	Angle of attack, sideslip, & paths				
19	Magnetic compass				
20	Latitude, longitude, & altitude				

*Figure 10: Fields Selected for GPS Output*

As the data is encoded in a binary format , it looks a little hostile on the wire.

*Figure 11: X-Plane GPS Data on the wire*

As discussed earlier in this document, for every field selected, a separate UDP packet will be sent. Each Index ‘flow’ is able to be identified by the 6<sup>th</sup> byte in the payload.

# pyHWLink\_USB\_Reader

This module interfaces directly to USB hardware. It runs a loop looking for any deltas in button positions. If a delta is detected, only the changed button states are sent to Pri Node Input.

It also listens out for a request from Send\_Query\_To\_Inputs, on receipt of such a request, all button state is send to Pri\_Node\_Input. Button state delivery is staggered in 20mS intervals to reduce the likelihood of over whelming the target simulator. Both open and closed button state is sent.

Sends to	Port
PyHWLink_Pri_Node_Inputs	26027
UDP_Reflector	27000

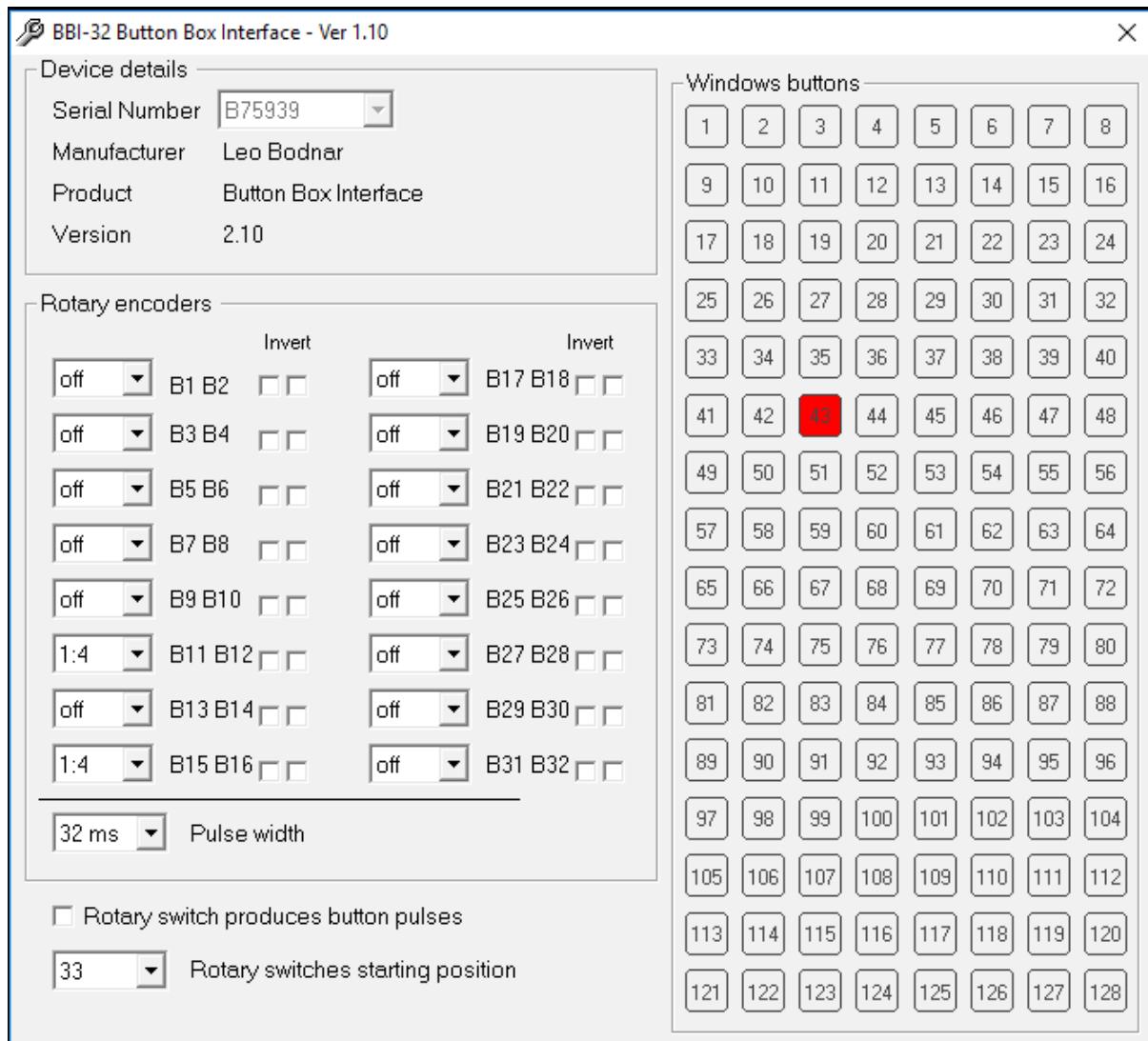
Listens to	Port
Send_Query_To_Inputs	

## USB Interfaces

There still is a need to use USB interfaces, such as for a High Resolution Analog interface (which already has noise reduction built in) or for Rotary Encoders. As the Leo Bodnar cards provide a solid rotary interface they will be used for inputs on radios and autopilots.

The follow vendor and product Ids are used for the Leo Bodnar cards:

	Vendor Id	Product Id
BU0836	0x16c0	0x05b5
BU0836X	0x1dd2	0x1001
BBI-32	0x1dd2	0x1150



The BBI-32 supports a special adapter for rotary switches that uses only two pins to monitor 12 inputs. This considerably reduces the amount of cabling needed to track rotary switch inputs. Up to 11 of these can be used on a single BBI-32, and they must be connected to the first 11 inputs. The BBI-32 configuration software enables you to determine at what port number the additional ports will start appearing.

Rotary encoders can be connected to any two sequential inputs. The BBI-32 rotary encoder configuration defaults to 80mS which means the minimum time between pulses is 80mS. Need to decrease to get a faster response time.

Needed to change the Rotary encoder from 1:1 to 1:4 to stop immediate echo as well as increase Pulse width to 32ms

The USB interfaces will still be normalised over the UDP interfaces, and should support requests to report the status of all Joystick interfaces, perhaps with a 10mS delay between the sending the status of each button input.

It appears that different Operating Systems provide different levels of accessibility, especially around devices they are Human Interfaces (HIDs). The Mac won't let you easily attach to the HID. This isn't too much of an issue, but does mean testing needs to be done on the Pi itself.

## SimConnect\_To\_IP

Currently SimConnect\_To\_IP is functional but can do with a clean up, and probably a rename. It currently sits outside the folder where the bulk of the python code exists. The folder is 'bne-arduino-flight-simulator-interfaces\SimConnectExtractor'. The project is called SimConnect\_to\_IP.sln

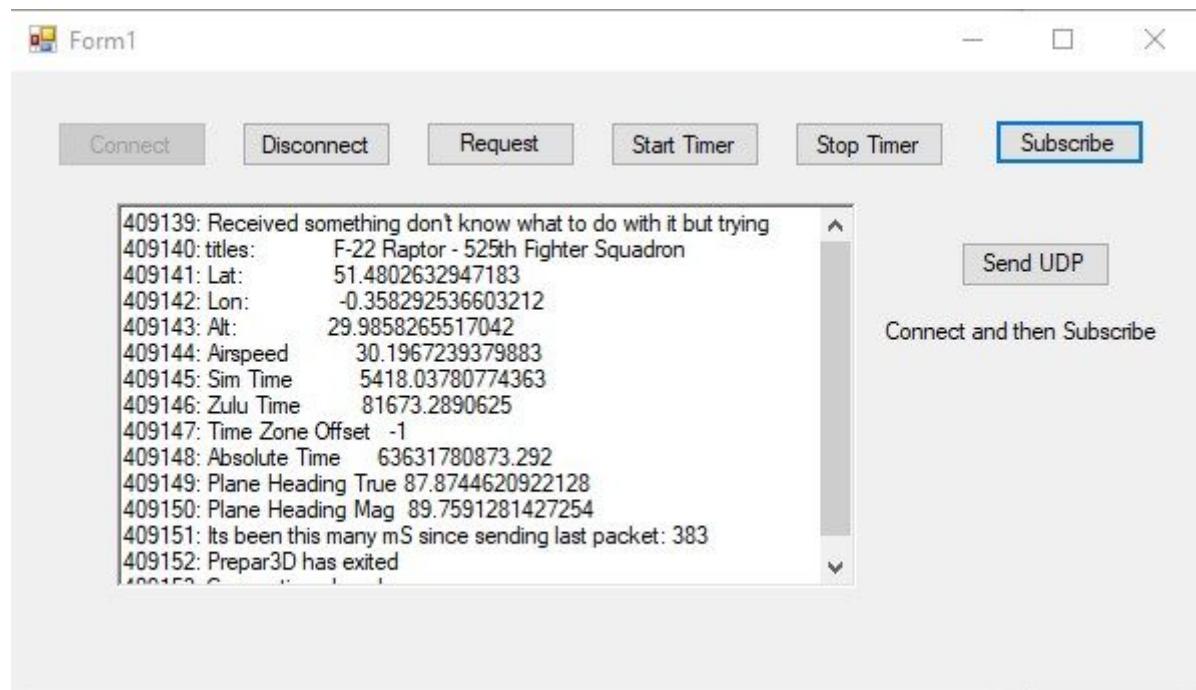


Figure 12: SimConnect\_To\_IP GUI

As of Jan 28 2019 there are a number of buttons that aren't needed. All that is needed to operate the code is to select connect, and then subscribe. As the button suggests, SimConnect\_To\_IP will then subscribe to a data stream from P3d. This is more efficient than polling P3d, as you are setting up structures and allowing P3d to update on a time base it expects.

Sends to	Port
pyHWLink_Serial	13136

Listens to	Port
P3d SimConnect	SimConnect API

```

25 2.583420      192.168.1.138    192.168.1.135    UDP    245 51860 → 13136 Len=203
20 3 107786      192.168.1.138    192.168.1.135    UDP    246 51860 → 13136 Len=204
<               ↳

0000 b8 27 eb 00 e2 14 34 e1 2d a0 b4 73 08 00 45 00 . '....4. -..s..E.
0010 00 e6 55 f9 00 00 80 11 00 00 c0 a8 01 8a c0 a8 ..U..... .....
0020 01 87 ca 94 33 50 00 d2 85 45 6c 61 74 69 74 75 ...3P.. Elatitu
0030 64 65 3a 35 31 2e 34 37 38 38 30 30 37 35 35 36 de:51.47 88007556
0040 39 32 37 2c 6c 6f 6e 67 69 74 75 64 65 3a 2d 30 927,longitude:-0
0050 2e 34 30 36 32 38 32 35 38 37 30 31 33 36 35 37 .4062825 87013657
0060 2c 61 6c 74 69 74 75 64 65 3a 32 35 2e 32 36 32 ,altitud e:25.262
0070 34 32 31 34 35 34 32 39 32 34 2c 61 69 72 73 70 42145429 24,airsp
0080 65 65 64 3a 31 33 30 2e 39 39 37 37 34 31 36 39 eed:130. 99774169
0090 39 32 31 39 2c 7a 75 6c 75 74 69 6d 65 3a 38 31 9219,zulu utime:81
00a0 35 37 35 2e 33 34 33 37 35 2c 74 69 6d 65 7a 6f 575.3437 5,timezone
00b0 6e 65 6f 66 66 73 65 74 3a 2d 31 2c 74 72 75 65 neoffset :-1,true
00c0 68 65 61 64 69 6e 67 3a 38 36 2e 39 35 36 38 34 heading: 86.95684
00d0 39 36 30 38 39 34 35 34 2c 6d 61 67 68 65 61 64 96089454 ,maghead
00e0 69 6e 67 3a 38 38 2e 38 36 30 35 39 34 30 38 35 ing:88.8 60594085
00f0 33 35 37 32 3572

```

Figure 13: SimConnect\_To\_IP Data on the wire

## PyHWLink\_Control\_XPlane

Whilst extracting information from X-Plane was relatively straight forward, trying to control X-Plane was a little more exciting.

Was originally trying to use the 'inverse' of data extraction by writing to an offset with a set of parameters.

```
('DATA'.encode('utf-8'), 0, 14, 0, -999, -999, -999, -999, -999, -999)
```

This didn't work so well. However found some very useful information at

<http://blog.shahada.abubakar.net/post/linux-udp-network-client-library-for-x-plane-10-and-11>, specifically in the section

### CMND (Client to X-Plane)

- char [5] name = CMND\0
- char[] cmnd = "sim/flight\_controls/flaps\_down"

This references a document in the X=Plane folder which explains how to control X-Plane, for X-Plane 11.26, it is found in X-Plane → Instructions → X-Plane SPECS from Austin → Exchanging Data with X-Plane.

The commands that are available to sent can be found :

To see the (plethora) of X-Plane commands, run X-Plane and go to the Settings menu, Joystick and Equipment screen, Buttons:Advanced tab.

The commands are the group name you see in the center of the screen, PLUS command string in the right side, all run together.

This didn't seem to be there in 11.26, but if you go Keyboard → All and then over over something if interest, you see the required string.

Once you have the needed string it needs to be assembled, packed and sent. The payload header is CMD, followed by a 0 (need to work out what that is), and then the string to be send.

```
values = ('CMND'.encode('utf-8'), 0, 'sim/flight_controls/flaps_down'.encode('utf-8'))  
packer = struct.Struct('4s B 32s')  
packed_data = packer.pack(*values)  
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM) # UDP  
sock.sendto((packed_data), (UDP_IP_Address, UDP_Port))
```

During initial testing – not all values operated. Inc-Flaps-Up, Inc-Flaps-Down. Wheels-Brakes-On appeared to operate at start of flight, not not after flight. Sim Pause and Quit operated ok, Gear-Up, Gear Down, Landing Lights didn't operate. This was due a slightly lazy copy of the initial config which has a fixed length for the string.

After correcting that – landing lights and brakes not operating correctly. Changed from brakes\_maximum to brakes\_regular (which mapped to keystroke B)

Lights was due to another copy and paste mistake, second part of string was flight controls when it should have been lights.

sim/flight\_controls/landing\_lights\_on  
sim/lights/landing\_lights\_on

Assigned a keystroke to the hold brakes regular, what this seems to do is require keystroke to be held, so for the moment will avoid hold commands.

Sends to	Port
X-Plane	49000
UDP_Reflect	27000

Listens to	Port
PyHWLink_Pri_Node_Inputs	TBA

## UDP\_Input\_Control

Based on OverPro's Joystick controller, this reads a matrix of switches and sends changes over UDP. As this is running on a Arduino Mega – it is not coded in Python. Currently it is used just for digital inputs, but could be extended to also monitor analog axis, sending deltas as the input move, just need to consider what 'smoothing' algorithm could be used.

Sends to	Port
PyHWLink_Pri_Node_Inputs	26027
UDP_Reflector	27000

Listens to	Port
Send_Query_To_Inputs	7788

OverPro's original code supported 256 buttons, in a 16 \* 16 Matrix on a Arduino Mega (2560 R3), but due to the Ethernet Shield pulling Columns 13 and 14 to 1.5V at all times the Matrix is 16 \* 14. As of 21/4/2019 had issues with Column 12 (counting from 0 – which may explain 13 above) and above not reporting changes. Column 12 cases Arduino to crash)

Digital Pins 22- 37 used as row0 ~ row 15,  
 Digital pin 38-53 used as column 0 ~ 15,

Even though internal pull-ups are enabled, found that the addition of 10K pull ups reduced noise induced transitions.

## Code Approach

On initialisation the pin modes are set, the arrays holding past and current switch state are initialised to 0. Currently a UDP ping is send out to the reflector in initialisation.

The code then enters a loop where all rows are individually pulled down. After a row is pulled down, there us a predetermined delay, after which the column results are read. The column results are then stored in an array (joyReport.button).

After checking if the indicator led should be transitioned, a comparison is made of past and future state. During this loop, any changes are individually sent out over UDP to both the Reflector and the Input Processing module.

## UDP\_Reflector

Displays and optionally forwards packets to a Wireshark receiver. Intended to be 'always on', acting basically as a sink hole until needed for troubleshooting. By having all IP enabled endpoints sending a couple of their payload to UDP reflector, packet monitoring is turned on at a single point instead of across all code modules.

By default displays all packets but will filter using simple character matching (no wild

carding)

Listens to	Port
All	27000

Command line options

- a: filterstring - display packets containing this string
- b: IP Address and Port to send to Wireshark

## Code Status

	Complete	Cmd_line	Debug	Config_File	Doc
pyHWLink_Serial	No				
pyHWLink_USB_Reader					
UDP_Reflector					

## Coding Learnings

## Logging

The logging module (import logging) enable you to determine which messages will be

displayed on the console

```
logging.basicConfig(format='%(asctime)s:%(levelname)s:%(message)s', level=logging.INFO)
#logging.basicConfig(format='%(asctime)s:%(levelname)s:%(message)s', level=logging.DEBUG)
```

Once the value is set it cannot be changed mid-execution, so it is best to set at the top of the program.

## Command Line Parameters

Command line parameters (from optparse import OptionParser) are easily managed with this module.

```
D:\>UDP_Reflector.py --help
2019-01-13 07:49:41,712:INFO:Unable to find UDP_Reflector_config.py
Usage: UDP_Reflector.py [options]

Options:
  -h, --help            show this help message and exit
  -w opt_W_Host, --wh=opt_W_Host
                        Wireshark Target IP Address
  -u opt_W_Port, --wp=opt_W_Port
                        Wireshark Target Port. 27001 is used if not explicitly
                        specified
```

Command line options are easily added, including help.

```
parser = OptionParser()

parser.add_option("-w", "--wh", dest="opt_W_Host",
                  help="Wireshark Target IP Address", metavar="opt_W_Host")
parser.add_option("-u", "--wp", dest="opt_W_Port",
                  help="Wireshark Target Port. 27001 is used if not explicitly
specified",
                  metavar="opt_W_Port")

(options, args) = parser.parse_args()

logging.debug("options:" + str(options))
logging.debug("arguments:" + str(args))

if options.opt_W_Host != None:
    wireshark_IP_Address = str(options.opt_W_Host)

if options.opt_W_Port != None:
    wireshark_Port = str(options.opt_W_Port)

if wireshark_IP_Address != None:
    logging.info("Wireshark host is : " + wireshark_IP_Address)
    logging.info("Wireshark UDP port is : " + str(wireshark_Port))

if len(args) != 0:
    filterString = args[0]
    logging.info("Display Filter is :" + str(args[0]))
```

# Configuration Files

A local file can hold settings for a program. Its a good idea to simple extend the name of the primary code with '\_config' at the end. The file type should be '.py'

e.g. PyHWLink\_UDP\_Reflector configuration file is UDP\_Reflector\_config.py

The values are loaded early in the code

```
try:  
    if not (os.path.isfile(config_file)):  
        logging.info('Unable to find ' + config_file)  
  
    else:  
        try:  
            from config_file import *  
  
        except Exception as other:  
            logging.critical("Error in Initialisation: " + str(other))  
  
            print('Unable to open "' + config_file + '"')  
            print('Or variable assignment incorrect - forgot quotes for string?')  
            print('Defaults used')
```

The configuration file looks like a series of variable assignments

```
learning = True  
AircraftType = 'A10C'
```

# Error Handling In Python

Its good form to use the try and except constructs,

```
def ReceivePacket():

    while True:

        try:

            data, (Source_IP, Source_Port) = serverSock.recvfrom(1500)

            ReceivedPacket = data
            packets_processed = packets_processed + 1

            ProcessReceivedString( str(ReceivedPacket), Source_IP , str(Source_Port) )

            logging.debug("Iterations since last packet " + str(iterations_Since_Last_Packet))

        except socket.timeout:
            iterations_Since_Last_Packet = iterations_Since_Last_Packet + 1
            if debugging == True and (iterations_Since_Last_Packet > 10000):
                print("[i] Mid Receive Timeout - " + time.asctime())
                iterations_Since_Last_Packet=0

            last_time_display = time.time()
            packets_processed = 0
            continue

        except Exception as other:
            logging.critical('Error in ReceivePacket: ' + str(other))
```

Of note in this code, is a workaround to avoid the use of threads. As the code has a single purpose of receiving a workload and then processing it, blocking isn't a big issue. However it is good form to provide some updates to the user

```
2019-01-13 08:25:39,410:INFO:Unable to find UDP_Reflector_config.py
Listening on port 27000
2019-01-13 08:25:44,424:INFO:Keepalive check 0 Packets Processed. 0 packets per second.
2019-01-13 08:25:49,438:INFO:Keepalive check 0 Packets Processed. 0 packets per second.
```

Having a catch-all exception handler ensures that something is logged if an exception occurs. Included in excepted captures should be a Ctrl-C to gracefully exit, cleaning up resources.

```
def CleanUpAndExit():
    try:
        # Catch Ctl-C and quit
        print('')
        print('Exiting')
        print('')
        try:
            serverSock.close()
        except:
            logging.critical('Unable to close server socket')
            sys.exit(0)

    except Exception as other:
        logging.critical('Error in CleanUpAndExit: ' + str(other))
        sys.exit(0)

def Main():
    print('Listening on port ' + str(UDP_Port))
    try:
        ReceivePacket()

    except KeyboardInterrupt:
        # Catch Ctl-C and quit
        CleanUpAndExit()

    except Exception as other:
        logging.critical('Error in Main: ' + str(other))

Main()
```

## Persisting Information

The two major design principles used in this project are only loosely couple (i.e. avoid maintaining state), and minimise the number of locations where unique configuration is stored. This approach keeps input modules simple, with only the primary input and output nodes needing to hold major pieces of configuration data.

Data is persisted in json, with the two major files being ‘input\_assignments.json’, and ‘temp\_input\_assignments.json’.

The following modules are used to manage data persistence.

```
from collections import OrderedDict
import json
```

*Drawing 1: Enabling JSON*

## Networking in Python

Sockets are the basis of the network stack in the project, with UDP being the preferred transport of choice due to its loose coupling.

With the following declaration the process will accept traffic from any host as we are using a '0' as opposed to a 127.0.0.1 which would only allow traffic from the local host. For Windows, it appears you can't simply use '0'. Also a very short timeout has been set, this means the programs can loop quickly

```
# UDP_IP_ADDRESS = "127.0.0.1"
UDP_IP_ADDRESS = "0"
UDP_PORT_NO = 26027

serverSock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
serverSock.settimeout(0.0001)
serverSock.bind((UDP_IP_ADDRESS, UDP_PORT_NO))
```

*Drawing 2: Network Stack Initialisation*

Once initialised we enter into a loop waiting for packets, and providing the user with a status updates when there are no packets being received. This approach may be changed once code has been completed, with perhaps a 1 second socket timeout.

```

def ReceivePacket():

    # a is used to track the number of timeouts between packets
    # throws a keepalive message to indicate we are still alive
    a=0
    while True:

        try:
            data, addr = serverSock.recvfrom(1500)

            logging.debug("Message: " + str(data))
            ReceivedPacket = data.decode('utf-8')
            logging.debug("Message: " + ReceivedPacket)
            ProcessReceivedString(str(ReceivedPacket))

            a=0

        except socket.timeout:
            a=a+1
            if (a > 100000):
                logging.info("Long Receive Timeout")
                a=0
            continue

        except Exception as other:
            logging.critical("Error in ReceivePacket: " + str(other))

```

*Text 3: Packet Receive Loop*

## Pick a Python Version

While documenting and testing the project, run into a surprise for junior hackers. The code running on the Raspberry Pi was exhibiting some subtle differences in behaviour. Originally I thought all code was running Python 3.5 or 3.7

But when running the Pri\_Node\_Input, on the PC or Mac, I wasn't capturing the informational updates associating with the Socket timeouts I was seeing on the Pi. Additionally I was seeing the PC asking for strings to be converting to byte objects, which wasn't seeing on the Pi.

The quick addition of the following to the code quickly answered the question:

```
import sys
print ('Running Python ' + sys.version)
```

The code running through the shell on the Raspberry Pi was reporting 2.7.13. Pi Idle3 was reporting 3.5.1 (not much of a surprise there...). Running from Thonny reported a 3.5.3.

On the PC through the explorer launched 3.7.2

To permanently change the preference on the Pi to run 3.5<sup>7</sup>

```
// Check to see if an alternative has been made
# update-alternatives --list python

# sudo update-alternatives --install /usr/bin/python python /usr/bin/python3.5 1

// Validate changes have been made
# update-alternatives --list python
```

Updating the default version of python immediately impacted the Pri\_Node\_Input module, loosing timeouts.

Added code to check for minimum version of Python (yeap it could be optimised)

<sup>7</sup> <https://linuxconfig.org/how-to-change-from-default-to-alternative-python-version-on-debian-linux>

```

MIN_VERSION_PY3 = 5      # min. 3.x version
if (sys.version_info[0] < 3):
    Warning_Message = "ERROR: This script requires a minimum of Python 3." +
        str(MIN_VERSION_PY3)
    print('')
    logging.critical(Warning_Message)
    print('')
    print('Invalid Version of Python running')
    print('Running Python earlier than Python 3.0! ' + sys.version)
    sys.exit(Warning_Message)

elif (sys.version_info[0] == 3 and sys.version_info[1] < MIN_VERSION_PY3):
    Warning_Message = "ERROR: This script requires a minimum of Python 3." +
        str(MIN_VERSION_PY3)
    print('')
    logging.critical(Warning_Message)
    print('')
    print('Invalid Version of Python running')
    print('Running Python ' + sys.version)
    sys.exit(Warning_Message)

```

After validating this noted a consideration difference in socket timeouts between Windows and the Pi, using the 'same timeout' windows reports every 15 seconds, whereas on the Pi it is reported every second.

## Processing Data

Data packets have a similar format that that of SOIC, with the first byte of the packet either having a 'D' for Data, or 'C' for commands. The received string is then split into records sets that are separated by a ','. Individual attributes are separate by a ':'

For the Pri\_Node\_Input module, the ability to dynamically assign a task in response to a given input. In the following code snippet a check is made to see if there is an existing assignment, if there no assignment, and the module is running in learning mode, an update is performed, otherwise a look up is make, and the resulting value is added the Values to be sent to the Si

```

# Switch is Opened
if str(workingFields[2]) == '0':
    if learning and input_assignments[workingkey]['Open'] == None:
        updateOpenAction(workingkey)
    print('Value for Open is : ' +
        str (input_assignments[workingkey]['Open']))
    if input_assignments[workingkey]['Open'] != None:
       .addValueToSend(str (input_assignments[workingkey]['Open']))

```

```

def ProcessReceivedString(ReceivedUDPString):
    global input_assignments
    global send_string
    global learning

    send_string = ""

    try:
        if len(ReceivedUDPString) > 0 and ReceivedUDPString[0] == 'D':
            # Remove leading D
            ReceivedUDPString = str(ReceivedUDPString[1:])
            logging.debug('Checking for correct format :')

            workingSets =''
            workingSets = ReceivedUDPString.split(',')
            logging.debug('There are ' + str(len(workingSets)) + ' records')
            counter = 0
            for workingRecords in workingSets:
                logging.debug('Record workingRecord number ' + str(counter) + ' ' +
                             workingRecords)
                counter = counter + 1

            workingFields = ''
            workingFields = workingRecords.split(':')

            if len(workingFields) != 3:
                logging.warn('WARNING - There are an incorrect number of fields in: ' +
                            str(workingFields))
            elif str(workingFields[2]) != '0' and str(workingFields[2]) != '1':
                logging.warn('WARNING - Invalid 3rd parameter: ' + str(workingFields[2]))
            else:
                logging.debug('Stage 2 Processing: ' + str(workingFields))

            try:
                workingkey = workingFields[0] + ';' + workingFields[1]
                logging.debug('Working key is: ' + workingkey)

                logging.debug('Working Fields for working key are: ' +
                             str(input_assignments[workingkey]))

                logging.debug('The value is: ' +
                             str(input_assignments[workingkey]['Description']))

                print('Value for Description is : ' +
                      str (input_assignments[workingkey]['Description']))

                # Switch is Closed
                if str(workingFields[2]) == '1':
                    print('Value for Close is : ' +
                          str (input_assignments[workingkey]['Close']))
                    if input_assignments[workingkey]['Close'] != None:
                       .addValueToSend(str (input_assignments[workingkey]['Close']))

                # Switch is Opened
                if str(workingFields[2]) == '0':
                    print('Value for Open is : ' +
                          str (input_assignments[workingkey]['Open']))
                    if input_assignments[workingkey]['Open'] != None:
                       .addValueToSend(str (input_assignments[workingkey]['Open']))


```

## Working with Datasets

As some of the modules (e.g. Pri\_Node\_Input) deal with largish datasets that should be persisted. The first design choice was which file format should be used, JSON was chosen

for readability and ability to update in other programs.

The second decision point was what Python construct should be used to retrieve, update and store these datasets. The Ordered Dictionary was chosen.

### Declarations

```
from collections import OrderedDict
import json
```

### Load

```
print('Loading Input Assignments from: "' + input_assignments_file + '")'

try:
    input_assignments = json.load(open(input_assignments_file))
```

### Save

```
def save_and_reload_assignments():
    # Save out to a temporary file and reload to ensure it is in shape
    global input_assignments

    try:

        json.dump(input_assignments,
                  fp=open(temp_input_assignments_file, 'w'), indent=4, sort_keys=True)

        input_assignments = None

        input_assignments = json.load(open(temp_input_assignments_file))
```

### Update

```
input_assignments[workingkey] ['Description'] = wrkstring
```

## Working with Dictionaries of Dictionaries

The data set representing input assignments is multi-dimensional, in that a key points to another dictionary.

The dictionary is initialised with a simple declaration using curly brackets

```
# Empty the Dictionaries
OuterSavedValues = {}
InnerSavedValues= {}
```

Once the dictionaries are initialised values are assigned/read using square brackets. A simple dictionary is referenced by

```
LocalVar = MySimpleDict[value]
```

In the case of Pri\_Node\_Input we have a multi-dimensional array, so information is accessed using two sets of brackets.

```
input_assignments[workingkey]['Description']
```

So given the following piece of JSON

```
"01:001": {
    "Description": "Landing Gear",
    "KeyboardClose": null,
    "KeyboardOpen": null,
    "UDPClose": "sim/flight_controls/landing_gear_down",
    "UDPOpen": "sim/flight_controls/landing_gear_up"
},
```

If workingkey is “01:001” then LocalVar would be assigned “Landing Gear”

```
workingkey = "01:001"
LocalVar = input_assignments[workingkey]['Description']
```

To add an item to the ‘outer’ dictionary (input\_assignments in this example) you need to first create the inner dictionary and assign values to it. Here’s an example that was used to couple entries from an Element Tree to a Dictionary.

```
# Create Inner Dictionary Entry
InnerSavedSGACLS = {}
InnerSavedSGACLS['Name'] = subelem.attrib['name']
InnerSavedSGACLS['Description'] = subelem.attrib['description']
InnerSavedSGACLS['ACL'] = newroot.find('ACLContent').text

# Create Outer Dictionary Entry by assigning Assign Inner Dictionary Entry
OuterSavedSGACLS[subelem.attrib['id']] = InnerSavedSGACLS
```

## Working with a GUI

One of the most common Cross Platform GUIs in Python is ‘tkinter’.

Basic starting code

```

from tkinter import *
root = Tk()
# Title bar text
root.wm_title("pyHWLink Lamp Output Emulator")

# Create the Canvas
canvas = Canvas(root, width=420, height=260)
canvas.pack()

# This is called from code just before mainloop

def tick():

    # Non selective clearing of canvas if needed
    canvas.delete(ALL)

    # Draw a 8*8 Matrix and randomly pick Red or Black

    for x in range(0,8):
        for y in range(0,8):
            if (random.randint(0,1) == 1):
                canvas.create_rectangle(50 * x, 30 + y * 30, 52 + 50 * x,
                                       62 + y * 30, fill='red')
            else:
                canvas.create_rectangle(50 * x, 30 + y * 30, 52 + 50 * x,
                                       62 + y * 30, fill='black')

    if timetoexit == 0:
        canvas.destroy()
        root.destroy()
        root.quit()
    else:
        canvas.after(100, tick)

# Do something - as it is event driven, if we aren't waiting for user input need to
# set a timer
# Refers to
canvas.after(1, tick)

root.mainloop()

```

# PI Addressing

## General Pi Configuration

As the internal network is isolated – IP Addresses are static assigned. To do this you can either manually edit /etc/dhcpcd.conf or right click on the wireless icon at the top right – and select Wireless and Wired Network Settings.

Then select interface of interest – for sim it is eth0 – with a setting of 172.16.1.2/24. Leave the other settings blank

```
pi@GenFrontPi:/etc $ ip -4 addr show | grep global
    inet 172.16.1.2/24 brd 172.16.1.255 scope global eth0
        inet 192.168.1.139/24 brd 192.168.1.255 scope global wlan0
pi@GenFrontPi:/etc $ cat dhcpcd.conf

interface eth0
static ip_address=172.16.1.2/24
static routers=
static domain_name_servers=
static domain_search=
pi@GenFrontPi:/etc $
```

## Huey Configuration

The Pi Ethernet interface has been configured with a secondary interface. This doesn't seem possible using dhcpcd.conf - so using /etc/network/interfaces. Using the Wireless interface to assign a default route and DNS server.

Currently unable to SSH to 144 but can 145 (wired). Also if a ping is made to wired interface - the reply originates with Mac address of wireless

```
auto lo
iface lo inet loopback

auto eth0
allow-hotplug eth0
iface eth0 inet static
    address 192.168.1.145
    netmask 255.255.255.0

auto eth0:1
iface eth0:1 inet static
    address 192.168.3.100
```

```
netmask 255.255.255.0

auto wlan0
allow-hotplug wlan0
iface wlan0 inet dhcp
wpa-conf /etc/wpa_supplicant/wpa_supplicant.conf
```

## Interfacing to a GPS

The Lowrance 2000C supports a NEMA input over a serial interface. The GPS has traditionally been connected directly to the PC running FSX, driven by the native GPS interface in FSUIPC. As P3D, DCS, and X-Plane are now all in use, it was time to build a common interface to the GPS.

The Lowrance 2000 supports the following NEMA Sentences

GLL, RMC, RMB, GGA, GSA, GSV, APB

Currently Exporting - RMC, GGA and GSA

NEMA Sentences

- \$GPGGA - Global Positioning System Fix Data
- \$GPGSA - GPS DOP and active satellites
- \$GPRMC - Recommended minimum specific GPS/Transit data
- \$GPRMB - Recommended minimum navigation info
- \$GPGSV - GPS Satellites in view
- \$GPAPB - Auto Pilot

After been driven nuts trying to understand why the Lowrance GPS seemed to be displaying incorrect data went to the Digital Data screen - it also mismatched the data I was sending to it. discovered previous owner has DM enabled, not DMS and Datum selection was set to NA 1983.. This was resolved by setting Datum to WGS 84 and coordinate system using DMS, and now things align

Also worthy of note is Seconds should be sent as Decimal value and the GPS will convert to Degrees ie 50 beings 30

## The Development and Test Environments

Final testing of the system can be centrally monitored using TeamViewer for the Sim PC and VNC for the Pi. Suggest allocating 172.16.1.5 address for the wired address on the Mac.

Unit testing and development can be a little easier on a single platform, and as Python is portable this can be done on the Mac. As it need to receive traffic from the Arduinos the wired Address on the Mac is set to 172.16.1.2 (that normally of the Pi – don't forget to change it back later), and if you aren't removing the Pi and Mac from the Sim network – don't forget to shut the Pi down.

## The Sim PC – making it usable

### Remote Control

Previously VNC was my go to choice – it was flexible and easy to setup. However in later version, you are unable to connect direct to the host using an IP Address unless you are running the Enterprise license. It looks like it is in relatively common trend – just ended up using TeamViewer instead which works very nicely. It does require both the Sim PC and Mac (my development platform) to have a connection to the Internet.

For the Raspberry Pi – the VNC Server that comes with the Distribution still works very nicely, remembering you by default authenticate with the Pi username password (ie pi and your password).

### The Firewall

Generally the Windows Firewall gets in the way of doing stuff that is left of field (such as stopping ping) - so it is nice to turn the firewall off – without completely compromising the PC. That is part of the reason for having using two Ethernet cards in the approach, allowing differentiated security rules to apply.

The Ethernet card with access to the Internet is treated as untrusted, ie public with the firewall running, and ethernet card on the 172.16.1.X network is treated as private and fully trusted with the firewall disabled. Unfortunately the Windows GUI doesn't allow you to change an Adapters Public or Private attributes once assigned.

WARNING – This doesn't seem to persist – checking to see if this is only after a crash or after a restart.

So PowerShell it is to list interfaces and profiles:

```
Get-NetConnectionProfile - lists interfaces and their profiles
```

```
Name          : Unidentified network
InterfaceAlias : 172.16.1.X Network
InterfaceIndex : 9
NetworkCategory : Public
IPv4Connectivity : NoTraffic
```

```
IPv6Connectivity : NoTraffic  
Name          : Network 3  
InterfaceAlias : Home 192.168.X.X  
InterfaceIndex : 14  
NetworkCategory : Public  
IPv4Connectivity : Internet  
IPv6Connectivity : NoTraffic
```

To change an interface profile:

```
Set-NetConnectionProfile -InterfaceIndex <index number> -NetworkCategory
```

So based on the output from above

```
Set-NetConnectionProfile -InterfaceIndex 9 -NetworkCategory Private
```

## Troubleshooting If Data isn't being received

Run wireshark and see if packets are actually arriving at the PC. If the firewall is in action you will still see the packets in Wireshark

If a service is supposed to be running and you know its port, running the following from the command prompt (is this example looking for port 49000).

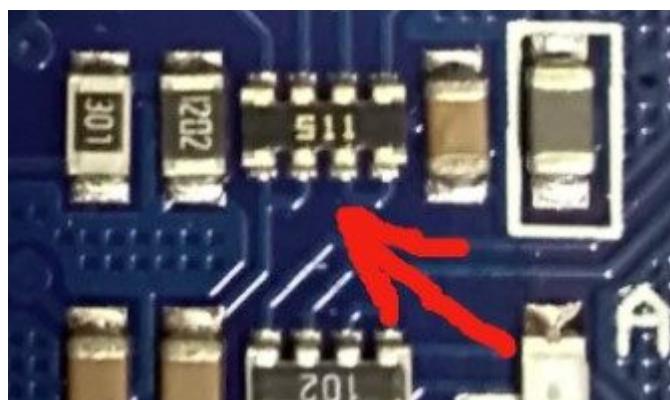
```
netstat -a | find "49000"
```

When working with DCS you could also log at adding log entries to the export.lua script when processing received packets.

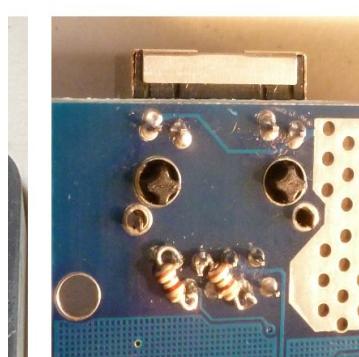
## Hardware Details

### Arduino Ethernet Shields - Attack of the Clones

If you are using cheap ebay Ethernet Shields there are a few things you may run into.



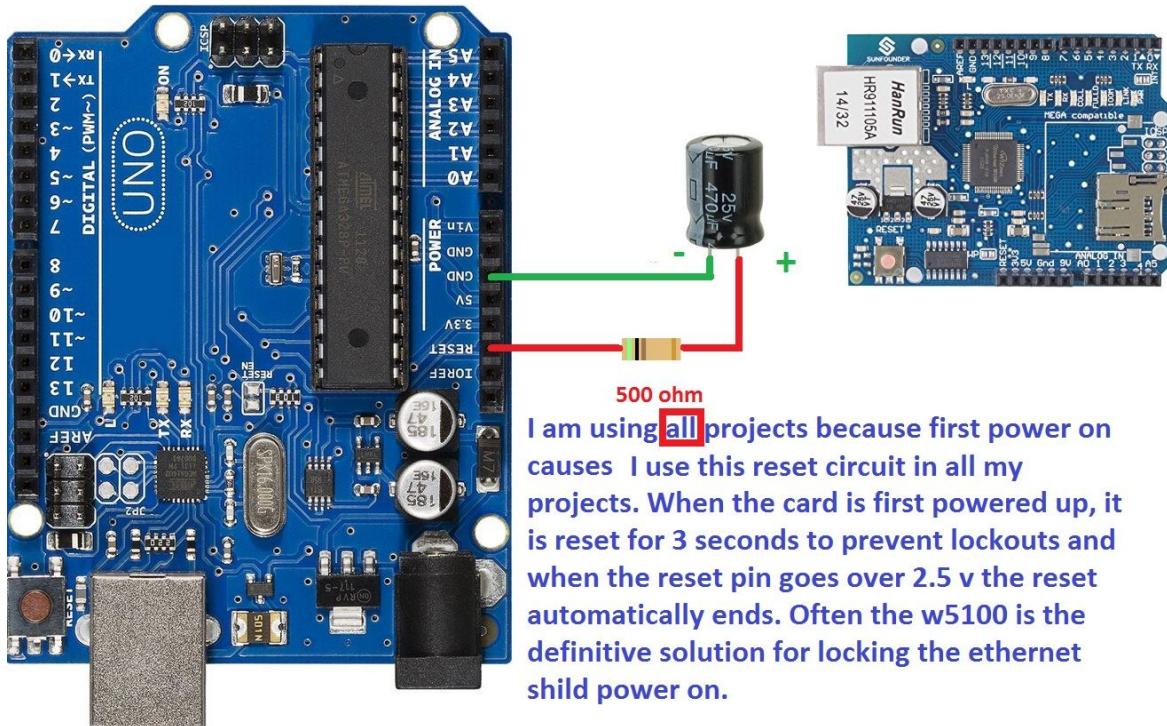
15



Probably the nastiest issue relates to incorrect resistor values between the 5100 chip and the RJ45 connector. According to the Arduino reference design the resistor array should be 51 ohms – but a number of the shields ship with 510 ohm resistor arrays.<sup>8</sup>

This can prevent the network switch/router bringing up the Ethernet link. It is solved by soldering a 100/220 ohm resistor between pins 1&2 and 3&6.

A second commonly encountered issue is the Ethernet shield not booting correctly on power up. This is a timing issue which is resolved by using a resistor and capacitor to extend the power on reset time on power up<sup>9</sup>. I used 220ohm and 200uF.



Before finding the fixes above I did find a fix which involved surgery of the shield. Basically the fix turned off speed and duplex negotiation by disconnecting pin 65 from the pcb. Whilst this enabled some of my shields to bring the link up it didn't fix all of them. The

<sup>8</sup> 511 instead of 510 ohms <https://forum.arduino.cc/index.php?topic=351477.30>

<sup>9</sup> <http://tigawd.blogspot.com/2015/05/arduino-uno-clone-and-w5100-ethernet.html>

resistor fix above was by far the most reliable.

#### OPERATION CONTROL MODES (65, 64,63)<sup>10</sup>

000 Auto-negotiation enable with all capabilities
001 Auto-negotiation with 100 BASE-TX FDX/HDX ability
010 Auto-negotiation with 10 BASE-T FDX/HDX ability
011 Reserved
100 Manual selection of 100 BASE-TX FDX
101 Manual selection of 100 BASE-TX HDX
110 Manual selection of 10 BASE-T FDX
111 Manual selection of 10 BASE-T HDX

10

[https://www.sparkfun.com/datasheets/DevTools/Arduino/W5100\\_Datasheet\\_v1\\_1\\_6.pdf](https://www.sparkfun.com/datasheets/DevTools/Arduino/W5100_Datasheet_v1_1_6.pdf)

## Building the Panels

There are two files used for the Hornet Build – Convert-Cad-In-Progress.vsd (based on free plans) and Hornet\_v49.vsd. The later file holds the larger panels and original indicators used in earlier projects.

One thing done differently in this build is the panels are attached to a larger Acrylic panel, this removed the need for Duz rails and fasteners and meant there was a repeatable process.

The tub of the Hornet pit is one of the early shells made by Dog Boss – with a flute plastic frame and printed cardboard over the frame. This provides a great starting point for the project. The cardboard for the panels was removed, measured, drawn up in Visio, and then test printed on two A3 sheets taped together. The Drawing Page size was set to a custom 508mm \* 304.8mm, No Scale (i.e. 1:1) and the printer page size set to Hobby V5 20\*12. The Rear left panel was 490mm \* 246mm

This uncovered an issue where – the image didn't fully appear in the Vector cut. What was needed was the image size to be entered by clicking on the small square box with four holes and entering the dimensions in centimetres. As the drawing is not rectangular, a rectangular boundary box was added, and made a different colour so it is assigned a different layer and then set to 0 passes. A light blue was used for the outer box.

Individual panels are then copied from Convert-Cad-In-Progress.vsd to a temporary worksheet. The original CAD layers are removed. And then a layer is added (Master-Cutout), it is made the active layer, and a inner shape is drawn for the panel cutout. Be aware of space needed for LEDs and nuts that hold the three panels together. It is best to use a series of rectangles as they are square to the edges of the panel. Once the clearance has been drawn – assign the panel name to a text layer, colour it red, make the outline of the entire panel red, and resize the mount holes to 3.3mm, to allow them to be tapped. Remove all unwanted elements and layers. Run a test print on a piece of paper to check for clearances.

Now group the items together, copy them to the master sheet and position.

### Line Colours Used

Light Blue – Outer Bounding Box

Light Green – Outer Panel Cut

Red – Text and Outline of Panels

Black – Inner Panel Cuts

To mark paper 5% power at 100% speed. It is important to tape the paper to the honeycomb support so it doesn't move. Mark the outline and then carefully place the cut panel on

the outline.

To Cut through Black 3mm Acrylic – 60% Speed – 90% Power – 4 passes. The Laser power did drop a little starting at 8.2mA, finishing at 7.9mA, but the cut was complete.

## Open Hornet

Open Hornet is an Open Source Project for building a F18C.

Primary Web Site <https://openhornet.com/>

Git Repository <https://github.com/jrsteensen/OpenHornet>

Discord channel <https://discord.com/channels/392833351238811648/>

As of June 2020, the project is still work in progress, with the UIP largely complete.

While you are able to download individual files from the Github site itself, it is a much smarter idea to install GitHub desktop and then clone the Open Hornet files to your local machine. Github desktop will then ensure your local copy is kept up to date.

KiCad is used for PCB design, Solidworks for Mechanical Design. The project team produce stl files, which enables you to simply open them in your favourite 3d Slicer, and then print them. For other objects, which are typically flat, such as the Ribs, dxf files are produced. These can be open/imported into a number of programs including Visio. If you are importing a dxf file to Visio, you should first create a new sheet, and then import the dxf. This more reliably scales the drawing.

If you want to run ahead of the completed parts of the pit, you can use the components in the MCAD folder. As Solidworks does not have a free version for non-commercial users, and the Solidworks licenses are not cheap, an option is Fusion 360 which does offer a non-commercial version. You need to first upload the .sldprt file, and then it can be opened in Fusion 360.

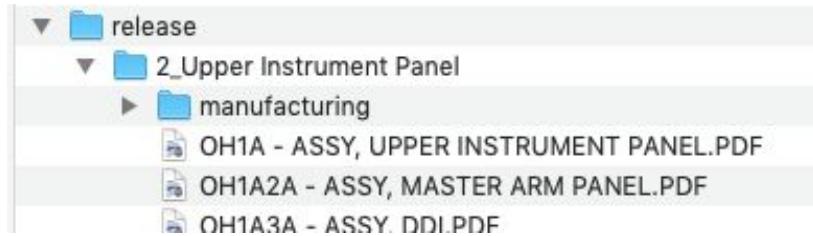
If you just want to look files, or get measurements, Solidworks offers a free viewing application, eDrawings. It is able to view sldasm (Solidworks Assembly) files, something I haven't yet seen work in Fusion 360.

### Major Folder Structure

ECAD	Electronic Design including PCB
MCAD	Mechanical Design
Release	Parts of the pit that are ready for using

## Finding your way around the release folder.

At first things may seem a little daunting, especially when you are presented with many files in a folder. The good news is there is order. If we look at the Upper Instrument Panel folder, we can see there are a series of PDFs and a manufacturing folder. The PDFs provide the assembly and painting instructions. The file names are important to note, start with the file with the lowest number of characters (prefix) at the front, eg OH1A - , this provides the top level design for Upper Instrument Panel. The prefixes get longer as it dives into sub-modules.



If we start with the top level pdf for the upper instrument panel, and go to the second page you'll find the all important parts list, this is the key to the files in the manufacturing folder.

PARTS LIST						
QTY.	PART NUMBER	DESCRIPTION	MATERIAL	MFR	TYPE	ITEM NO.
1	OH1A2A-1	ASSY, MASTER ARM PANEL	N/A	OPENHORNET	ASSY	1
1	OH1A7A-1	ASSY, SPIN RECOVERY PANEL	N/A	OPENHORNET	ASSY	2
1	OH1A8A-1	ASSY, UPFRONT CONTROL PANEL	N/A	OPENHORNET	ASSY	3
1	OH1A8A2A-1	ASSY, HUD PANEL	N/A	OPENHORNET	ASSY	4
2	OH1A3A-1	ASSY, DDI MODULE	N/A	OPENHORNET	ASSY	5
1	OH1A4A-1	ASSY, EYEBROW WARNING INDICATORS, LEFT	N/A	OPENHORNET	ASSY	6
1	OH1A6A-1	ASSY, EYEBROW WARNING INDICATORS, RIGHT	N/A	OPENHORNET	ASSY	7
7	OH1A-10	GLARESHIELD MOUNTING TAB, LEFT	N/A	OPENHORNET	ASSY	8
7	OH1A-11	GLARESHIELD MOUNTING TAB, RIGHT	N/A	OPENHORNET	ASSY	9
1	OH1A-100	UIP FLOOR	.75 MDF	OPENHORNET	CNC	10
1	OH1A-101	UIP FWD BULKHEAD	.75 MDF	OPENHORNET	CNC	11
2	OH1A-102	UIP LEFT OUTBOARD LONGERON	.50 MDF	OPENHORNET	CNC	12
1	OH1A-103	UIP LEFT INBOARD LONGERON	.75 MDF	OPENHORNET	CNC	13
1	OH1A-104	UIP RIGHT INBOARD LONGERON	.75 MDF	OPENHORNET	CNC	14
1	OH1A-105	UPPER HUD BOX SUPPORT	.50 MDF	OPENHORNET	CNC	15
1	OH1A-106	UIP FRONT PANEL FACE	.25 ACRYLIC	OPENHORNET	CNC	16
1	OH1A-107	UPPER UIP FACE MOUNT, LEFT	PETG	OPENHORNET	PRINT	17
1	OH1A-108	UPPER UIP FACE MOUNT, RIGHT	PETG	OPENHORNET	PRINT	18
2	OH1A-109	UIP DISPLAY PRESSURE FOOT	PETG	OPENHORNET	PRINT	19

Typically when you see a material listed, that an item to print, cnc, etc, as opposed to an Assembly which surprisingly enough has Assy in its description :) For the Assembly files, there should be a corresponding pdf.

From the list above, we find 'OH1A-100\_UIP\_FLOOR\_.75\_MDF.dxf' in the manufacturing folder, as well as 'OH1A -107 - UPPER UIP FACE MOUNT, LEFT.stl'

# Fusion 360 – Bodies and Components

A Body is the smallest piece to work with and must be continuous, a component can serve as a container for multiple things including bodies.

## Working with STL Files

I had need to get a cutting template for the Selective Jettison button.

To import the STL file – first create a new project in Fusion 360, optional create a sketch with a rectangle that bounds the STL file, and then Insert → Mesh Object. You can then orient the object relative to the plan.

At this point there isn't a lot you can modify the STL file. To convert it to a body<sup>11</sup>, first disable history – and then select the Mesh object, right-click and select Mesh To BRep.

It appears you cannot create a drawing from either the import STL file, or the BRep object. Found it was easiest to create a sketch and draw an outline against the object.

Generally buttons are solid, to add an insert to fit on the button

Import the Mesh

Convert to BREP

Create a new components

Paste the Bodies from Master project

Using a sketch that has the outline of the insert, extrude it to cut into the base of the button

Move the new component into the cavity that was created

I'd originally been printing with 20% infill, but not, if the object is not load bearing, using 0% infill, but always do enable bridging support when slicing. Moving from 20% down to 0% reduces a 6 hour print job to 4 hours (MFD Lower support).

## Working with DXF in Visio

DXF files are easily imported into Visio. As I don't have a large CNC, needed to break them down into smaller chunks

Start a new Visio diagram. Set sheet size to 20x12.

Import CAD – but clear the all of the protect check boxes.

11 <https://toglefritz.com/convert-a-simple-stl-into-a-body-in-fusion-360/>

The CAD object is likely to be largely that a sheet, so duplicate the sheet, place a line through a desired cut point duplicate again and then trim. Note you'll probably have a surrounding rectangle you can't see, but can select. Delete it. Remove the unwanted side, and then add a rectangle of known size, which is used to set the vector layer to the correct dimensions. See Panel cutting for tips on this (110)

On some drawings needed to specify custom scale which was 1mm to 1mm with CAD drawing units in Inches

## Graphical OLED – SSD1306

After a number of years of working with Character based OLED displays, the time finally came to expand coding horizons to add a Graphical Based OLED. The need arose from needing some hatching and different font sizes with the Hornet Altimeter.

Originally attempted to have the graphical code run alongside Stepper motor code on the Arduino, but my body code didn't co-exist, and as I wasn't up for learning multi-threaded code on Arduino, the code moved to the Raspberry Pi. The different generations of code are still in the Git Repo (well as of 20190311).

Some of the following is duplicated in the Pi code, but decided I'd prefer to see a single spot to reference hardware setup.

Altimeter OLED code - used in conjunction with Arduino for driving stepper

This code is based of the SSD1306 code developed by the adafruit team. Suggest starting with same code form Adafruit team to ensure hardware is correctly configured. The adafruit library will need to be grabbed from Github

<https://learn.adafruit.com/ssd1306-oled-displays-with-raspberry-pi-and-beaglebone-black/overview>

Then use this code.

Font used in Arduino build is FreeMonoBold which is basically Courier  
As the same font wasn't easily found, used 'monofonto.ttf' from

[http://www.dafont.com\(bitmap.php](http://www.dafont.com(bitmap.php)  
This font is fixed width, and digits are centered which make the dial look more natural  
The file needs to be copied to the directory where the python will execute from

This code assumes used on the SPI interface, which must be enabled in the pi  
To validate the SPI interface is active 'ls /dev/\*spi\*. This should return  
'spidev0.0 spidev0.1'.

The SSD1306

The following is the 'as wired' for the Arduino – noting that the SSD1306 is now connected to a Raspberry Pi to improve stepper performance on the Arduino. The SPI bus continues to be used. Of interest is the SSD1306 has options for SPI3 and SPI4 for addressing

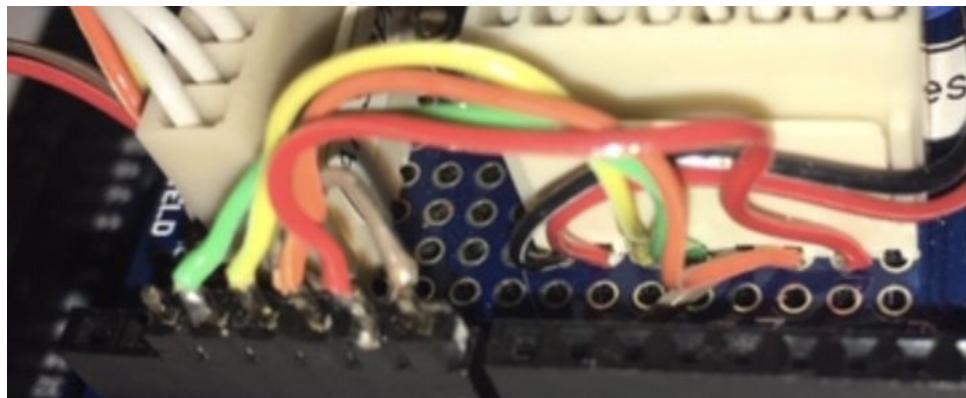
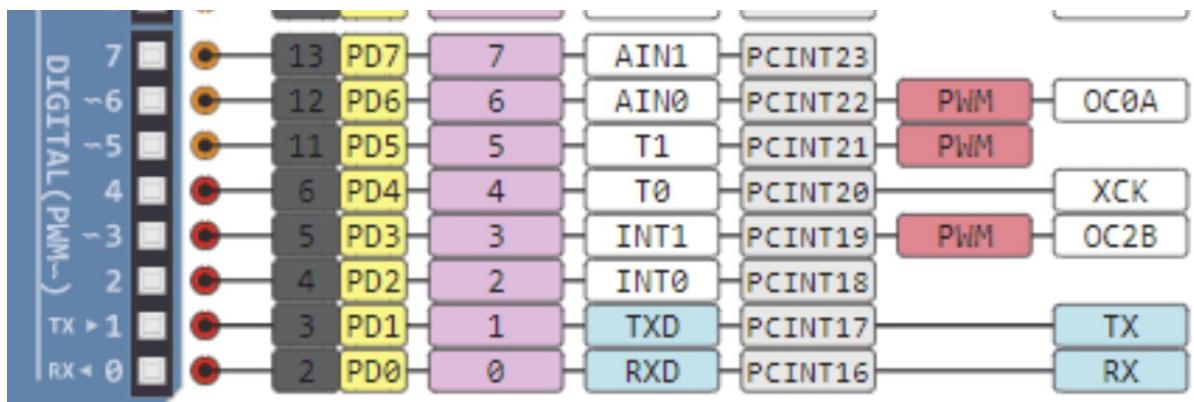


Figure 14: SSD1306 Arduino Wiring

SSD 1306 Pin		Color	Arduino Pin	Pi Pin	Pi Role
1	Gnd	Black		9	Gnd
2	Vdd	Red		1	3.3V
3	SCK D0	Yellow	PD-3	23	SPI_CLK
4	SDA D1	Green	PD-4	19	MOSI
5	RES	Brown	PD-5	18	GPIO 24
6	DC	Orange	PD-6	16	GPIO 23
7	CS	Red	PD-7	24	SPI_CEO
8					



MISO (Master In Slave Out) - A line for sending data to the Master device

- MOSI (Master Out Slave In) - The Master line for sending data to peripheral devices
- SCK (Serial Clock) - A clock signal generated by the Master device to synchronise data transmission.

## Building the Panels

### Panel Construction

The Panels in the pit are made of three layers of 3mm Acrylic, Base, Spacer, and Top.

The Base Layer is made of Black Acrylic, and has two primary roles, the mounting of switches and supporting the backlighting LEDs.

The Spacer Layer is also made of Black Acrylic, providing a clearance for the mounting nuts. It also has zones cut out to allow the backlighting through to the top panel. Black Acrylic is used to isolate light sources to reduce the likelihood of light bleeding to unwanted areas of the Top Layer.

The Top Layer is made of White Acrylic, and largely is used to hide the mounting nuts and provide the panel lettering.

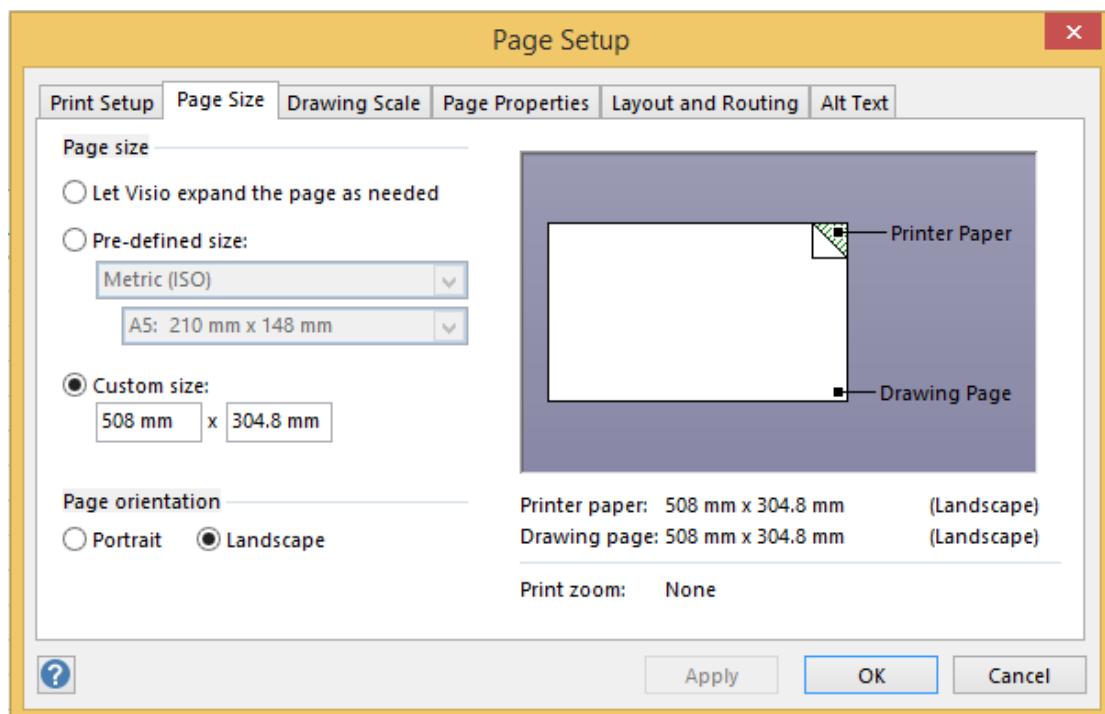
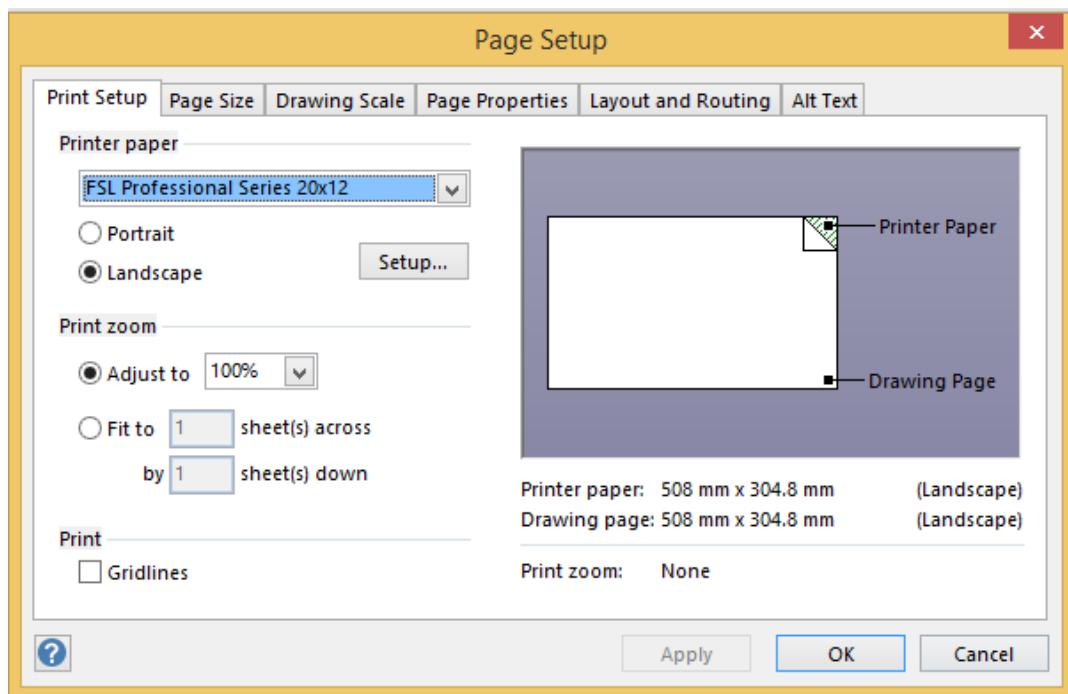
### Panel Design

The panels have been designed using Visio, more for historical reasons than anything else.

Source material for the panels has been a combination of photos, and in the case of the Hornet CAD diagrams provided on the DCS Forums site.

A separate Visio file is created for each aircraft, with a new sheet for each panel. The new sheet is duplicated from the Drawing Template. Copy the Panel from the Working CAD. **It is important that the CAD drawing is resized to the correct dimensions.** Generally panels are 145.45mm wide.

As some panels are larger than others, the default A4 paper size will no longer suffice. It is very important that the drawing paper size matches the printer size. Under page setup set as per the following (there is also a Hobby Series 20x12 which should really be used). Note the Drawing Page Size is explicitly set to 508\*304.8mm. The Drawing Scale should be None - ie 1:1



The Drawing Template sheet holds the following layers

LayerName	Layer Description
0 Base	Usually the largest of the layers The holes for mounting the entire

LayerName	Layer Description
	assembly to the pit are 4mm in diameter.
	The holes for linking the three layers together are 3.5mm in diameter.
	The holes for mounting the LEDs are 5mm in diameter. Try to avoid having the LEDs too close to the edge of the panel as it commonly results in very narrow edges of the spacer panel.
0 Base Text	This holds the name of the panel, its layer, and also a tag ('C') to be applied holes that need to be countersunk, such as the supports for the associated PCB.
1 Spacer	Cutouts for the mounting nights and backlighting. Clearances for the mounting holes and the holes linking the three layers together.
1 Spacer Text	This hold the name of the panel and its layer. Very infrequently the counter tag is applied.
2 Top	Cutouts for the switches and potentiometers and the mounting holes.
3 Text	The text that is to be engraved on the Top Panel
Component Outline	This provides the clearances needed for components along with the expected light coverage provided by the LEDs
Connector	TBA
Instructions	A set of basic checks to be done before printing.
Template - Corner	Prebuilt corners for mounting hole clearances

The Laser Settings sheet has recommended Laser Setting as well as a library of components with their associated hole cutouts for all layers and the component clearances.

The Drilling Templates sheet has layouts for mounting PCBs that are not normally associated with the panels.

## Merging Diagrams into Large Panel Diagrams

For this project, individual panels are in Convert-Cad-In-Progress (where the CAD diagram was imported). The primary Hornet Visio is Hornet\_v049, once github was used to host the project, the version numbers are no longer incremented.

- Duplicate a page from a previously converted panel.
- Delete the old diagram and rename the sheet to the new target panel
- Delete all layers – easiest way to do this is Remove Unreferenced Layers. Then add Master Outline Cut – make it the Active Layer.
- On the source document, ensure all layers are turned on, and then copy the diagram to be inserted.

- Paste the diagram into the new sheet, and if needed, drag the diagram so it fits on a single page
- Close the source document (Convert-CAD-In-Progress)
- Delete the Instruction Text Block
- Carefully Delete items on the CAD layers – note some items may incorrectly still be tagged with the CAD layers along with the desired layers, so double check.
- Once the noise has been reduced (ie some decluttering), draw the outline, generally the use of multiple overlapping triangles works best as everything is square.
- It may be easier to set the Master Outline Cut layer colour to Red (even though lines will still be Black).
- Once the outline has been determined, hide all layers aside from Master Outline Cut, select all lines, trim unwanted lines and join the outline together. Again check the outline is on Master Outline Cut
- Check there is suitable clearance our the head of the 4mm mounting bolts
- Add the 4mm mounting holes to the Master Outline Cut layer.
- Remove all other fragments so only the Panel Base Outline and Master Outline Cut remain, as a double check remove unreferenced layers.

## **Panel Pre Cutting Checklist**

- Label Base and Spacer with Unique Names – on separate layer so they can be rastered.
- Ensure there is adequate clearance for the mounting bolt heads on the base plate.
- 7mm rounding for indents for mounting
- When printing change Base/Spacer layer to Yellow. Print Base/Space Layer and Base-Text-Spacer text layers .
- Add PCB Support and holes to join panels.
- Mark Countersunk holes with a 'C'
- For curved corners, duplicate base frame and assign to spacer and top copy pieces AFTER TRIMING AND JOINING ADD 1MM RADIUS TO LINE to approx position and then align right/top etc
- Check LEDs aren't imposing on other components and there is suitable clearance from the panel edge
- Check Led Spacer holes are NOT HITTING LEDs

## **Panel Cutting**

Different cutting speeds and power levels are needed for different coloured Acrylic and different depths. Also as the laser has longer runs, it heats up, which results in a lower current passing through the tube. The drop in output power is partially mitigated by not setting the Laser Output Power to 100%, instead running it 85% seems to hold a more consistent output level.

Text is normally left as Black, with the Vector cut layers set to Yellow – that enables you to print text use Raster, and then select Yellow Layer for Vector cutting.

Colour	Colour Role
Black	For text as Raster looks at darkness of colour, not the colour itself For the larger panel – sub-panel work, this is the actual desired cut.
Yellow	For the layer that is to be cut. You do not need to change individual lines in Visio instead set the Layer to a colour.
Light Blue	The outer box when the diagram isn't rendering correctly in Retina Engrave. Capture the dimensions from Visio to set dimensions in Retina Engrave Set the Vector Layer to 0 passes.
Green	The outer shape of the large panel to be cut. Normally this will be first printed to a sheet of paper, and then the panel placed on it for the sub-panel cuts.
Red	The outline of the panel itself, normally just used for positioning on the larger pane.

If there is any doubt in the size or dimensions, do a dry run using paper (A3 sheets tapped together may be needed for larger panels). A speed of 100% with power level of 3% is good for marking, and a 100% speed with 13% power is needed for cutting out.

Larger jobs often are skewed in Retina Engrave 3D. Sometimes it is obvious, other times very subtle, which again is why printing anything that is in doubt on paper saves a bunch of time and materials.

If doing a test run on paper use three colours, one for the outer dimension box, the second for the outer of the shape itself, and then black for the remainder. Mark first @ 1%, and then cut paper @ 15%

The following steps will help

- Complete Drawing
- Draw outer box – noting its dimensions – make it a different colour to the rest of the diagram so it can be isolated to a different layer in Retina Engrave
- Ensure paper size and drawing size match 508 \* 304.8
- Don't bother importing Raster
- It is likely image will be incomplete but there should be no mention of image resized in right hand status box
- In the Vector view click on the box which is an outline with 4 holes (Resize according to input values)
- Enter the dimensions of the outer box (in cm)

- NOTE have lost registration when running a second cut – better to go for 4 cuts @60% speed 90% power
- There should only be a single copy of the drawing in Retina Engrave 3D.

After the cut is complete, a touch should be used to see if the ‘fins’ on the honeycomb can be seen.

## Documenting the Project

The project has been documented using LibreOffice, which has the benefit of running on MacOS, Windows, and Raspberry Pi. It behaves much like Microsoft Word, but is free.

There are some tricks worth knowing:

When including an image, change the Anchor to Anchor as a Character. The image can be cropped within the document.

Whilst the Word Processing has basic drawing capabilities, it is easier to use Drawing program and then save as an image, and import the image into the document. That ensures the diagram layout remains consistent. Diagrams for the Hornet project are generated in pyHWLink\_Diagrams.odg

## Sharing Progress Updates

To date most updates have been shared on ViperPit and DCS. With the Hornet project updates are being shared on HornetPits.org and DCS.

The F16 progress photos were hosted on PhotoBucket – but the site had a change of heart on funding – so now postimg.cc.

After uploading the progress photo to postimg.cc – reference it on the site – format for HornetPits

```
[url=https://postimg.cc/K15QCNGp] [img width=500 height=256]https://i.postimg.cc/x1w7JxWQ/Top-View.jpg[/img] [/url]
```

## System Build

All target systems should have git, LibreOffice and python3 installed. If Arduino code is to be developed, the Mac and Window’s environments support installation of the Arduino IE. The Windows environments needs the C# development environment, which typically is Visual Studio Community Edition.

Windows Build (as of 20190504)

Chrome

Free Download Manager 5 (Chrome extension enabled)

GitHub Desktop (1.6.5)

Python 3 (3.7.3)

LibreOffice (6.2.3.2)

Visual Studio 12

Acronis True Image 2017

Notepad++

Lockheed Martin p3d v2

DCS World 2.5.4

X-Plane 11 Demo

20190615 – added VNC viewer show Pi work can be done from Sim computer (no mouse and keyboard swapping)

It appeared post Visual Studio installation, p3D v2 errored with missing mfc100.dll. After trying a range of things finally resolved with Microsoft Visual C++ 2015-2019 Redistributable (x86) – 14.20.27508.

Download Visual Studio developer pack 4.72 (base reference for projects)

Enabled Community license

Resized Primary Screen size to bit Soniq Monitor (1804 \* 1004)

Adjust Power Savings to Never Turn off and PC, and Sleep the screen after 25 minutes.

Linked PC to MS Account

Disabled One Drive (removed in Optional Windows Components – which had done a nasty and moved desk to a OneDrive directory – which of course was deleted

Changed UAC never to notify to stop alters when launching p3D and DCS.

## **Hardware To Do's**

PCB for I2C mux using Adafruit card (UFC and A10)

PCB for Power Distribution

PCB for Lamp (need to get lamp dimensions)

## **Code To Do's**

Font to emulate 14 Segment display – probably easier to use a font designer

## **pyHWLink\_USB\_Reader**

1. Add Network stack for both sending state, as well as listening for Send\_Query
2. Add error handler
3. Add Standardised logging
4. for UDP\_Input add code to respond to send all input state (ie respond to a CQ)

## **Index of Tables**

DCS LUA Initialisation - Code Export.lua.....	21
DCA LUA Event Code - Export.lua.....	22
Packet Receive Loop.....	34

## **Index of Tables**

X-Plane - Selecting Indexes to export.....	5
X-Plane GPS Indexes selected.....	6
X-Plane Joystick showing commands.....	7
X-Plane Keyboard showing commands.....	7
System Flows.....	13
Detailed System Flows.....	14
PhyHWLink_GUI_Sender GUI.....	17
PhyHWLink_Pri_Node_Input User Interface.....	20
Fields Selected for GPS Output.....	23

X-Plane GPS Data on the wire.....	23
SimConnect_To_IP GUI.....	24
SimConnect_To_IP Data on the wire.....	25