Python HW Link

High Level Overview

As Flight Simulators increasingly offer support for API interfaces (eg 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 as 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.

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

* Used 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 impacted other modules.
* Accept inputs from push, toggle, rotary, and rotary encoders.
* Outputs – analog and digital, text. All outputs are normalised before being send 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 checkerboard 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

# 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).

As the Arduino doesn’t