A Multi-Raspberry Pi Organ

Designed and built by Ian Shatwell

Introduction

I have interests in computing, music and wood-working, and this project combines all three, with the intended end result of building a chamber organ: small enough to fit in a normal room, yet still having multiple consoles and a reasonable selection of stops. A replica of the Royal Albert Hall organ, above, with four manuals, pedals, 147 stops, numerous couplers and presets, and 9997 pipes, each weighing anything between a few grams to about a ton, is probably excessive, not to mention expensive, but two manuals with pedals, twenty or so stops and only virtual pipes should be achievable within the space and budget available.

I am an occasional church organist and have had the chance to study the local church (electric) organ while renovating the amplification stage. One of the other potential options was to MIDI-fy the console and add a modern computer based sound production unit to replace the current system built around electronic oscillators and filters. Electronics from the 1980s are very robust by modern standards, but do die eventually, so this may still be a future requirement.

Making a full wind-powered organ, along the lines of the this one or this, would be an interesting challenge, but the pipework and bellows require a lot of room and that much wood is expensive. This project will involve constructing a 'Virtual Pipe Organ', or VPO, which uses a computer to interpret signals from the console, and synthesize the sounds. High quality commercial software is available to do this (e.g. Hauptwerk) and can use any MIDI controller as an input. Two standard MIDI controller keyboards on a staggered stand, a MIDI capable pedal board, a high-spec computer and a software licence, and everything is done for two or three thousand pounds. But where is the fun in that? There are also good reasons for MIDI not to be used in an organ, especially when restricted to the original serial baud rate, so this design will attempt to avoid it when possible.

Initial tests

A [Piano HAT](#) from [Pimoroni](#) provided a suitable test platform for determining whether the RPi could generate suitable sounds. Having had previous experience with using an early RPi for an audio player the built-in audio was skipped entirely in favour of a cheap C-Media CM108 based USB 'sound card' device, which was then later replaced by a Behringer UCA222 module. Audio quality through the built-in port has been improved since the early Pi days, as has the quality of available USB or HAT based add-ons. The SunVox and Yoshimi synthesizers are supported by the Piano Hat supplied software, but TiMidity also works and just needs to be added to the list of supported synths to be recognised. From later experience it is likely that FluidSynth would have worked too, although the name to be added into the support list would be just 'Synth'. Success with this proved the potential for a hardware switch collection to be monitored, translated into sound generation commands (MIDI in this prototype test case) and then the required sounds to be produced with an acceptable quality.

Alternatively the following testbed code plays through the full MIDI range on whatever suitable device it can find.

```
#!/usr/bin/python

import sys
import pygame
import pygame.midi

# Initialise pygame
pygame.init()
pygame.midi.init()

# Suitable devices
supported =
['yoshimi','SunVox','TiMidity','Synth']

# list midi devices
devcount = pygame.midi.get_count()
suitabledev = -1

print "\n{} midi devices
found".format(devcount)
```

```
print "Device\tInput?\tOutput?"
for m in range ( 0, devcount ):
  devinfo = pygame.midi.get_device_info(m)
  devname = devinfo[1].split()[0]
  inputdev = devinfo[2] == 1
  outputdev = devinfo[3] == 1
  inuse = devinfo[4] == 1
  print "{}\t{}\t{}".format(devname, inputdev,
outputdev)
  if suitabledev <0:
    if devname in supported and outputdev and
not inuse:
      suitabledev = m
if suitabledev >= 0:
  print "\nUsing
{}".format(pygame.midi.get_device_info(suitabl
edev)[1])

if suitabledev < 0:
  print "Unable to find a suitable midi
```

```
if devcount<1:
  sys.exit(1)
```

```
device"
  sys.exit(2)

midiout = pygame.midi.Output(suitabledev,
latency=0)

for n in range (0,127):
  midiout.note_on(n, 127,0)
  pygame.time.wait(50)
  midiout.note_off(n)
```

Test_Midi.py

This testbed requires pygame to be installed, which is not used for the final organ, but it is then useful for trialling different software synths. Setting up sound on a Linux platform can be obscure. Using a CM108 based USB soundcard, which appears in the listing of '`aplay -l`' to have a device name of just 'Device', the following configuration works. The Behringer UCA222 configuration replaces 'Device' with 'CODEC', but is otherwise the same. TiMidity also may require a soundfont specific configuration. See the man page for details, or some of the soundfont sites include the required files.

Edit or add `dtparam=audio=off` to /boot/config.txt to disable the on-board audio.

Add `snd-seq-midi` to /etc/modules-load.d/modules.conf.

Comment out `options snd-usb-audio index=-2` in /lib/modprobe.d/aliases.conf

Edit or create /etc/asound.conf

```
pcm.!default {
  type hw
  card Device
}

ctl.!default {
  type hw
  card Device
}
```

To start fluidsynth running directly on the Alsa sound system:

```
fluidsynth --server --no-shell --audio-driver=alsa -o
audio.alsa.device=hw:Device /usr/share/sounds/sf2/eorg104a.sf2 &
```

Alternatively, to start timidity on top of jackd:

```
jackd -R -P70 -p16 -t2000 -dalsa --device hw:Device -P -p128 -n3
-r44100 -Xraw &>/tmp/jackd.out &

timidity -iAD -B2,8 -Oj --cache-size=4194304 --no-unload-instruments
-A50 --volume-curve=1.3 --config-file=/etc/timidity/timidity.cfg &
```

Trials with various soundfonts were also run at this point. Jeux, and English Organ both worked well, although Jeux proved to be too complicated for a RPi to handle once heavier demands were put on it.

MIDIorgan and Stratman Virtual Instruments also have a number of other soundfonts available which are likely to work.

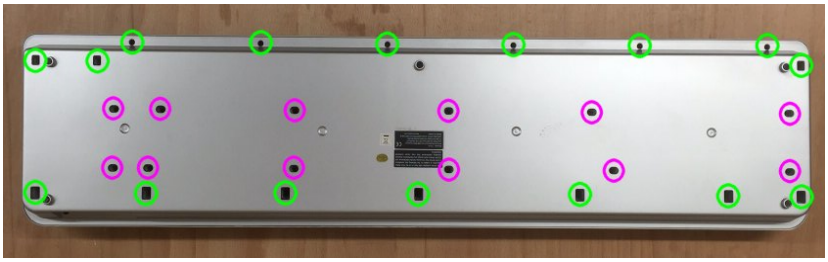
Keyboard selection

A number of reasonably priced midi controller keyboards are available. A standard organ manual has 61 notes (five octaves, C to C), which is conveniently one of the standard sizes and ranges of MIDI controller keyboards. These are available, new, from about £65, but at this price they often have unweighted 'synth style' keys. The M-Audio Keystation 61es keyboard has the advantage of being 'semi-weighted', and used to be reskinned by a local organ maker (Romsey OrganWorks) for making their VPOs, so has been proved to be workable. However, they kept the full on-board MIDI processing rather than stripping it back to the bare note switches. This particular keyboard now seems to have been superseded by the Keystation 61 II, but is frequently available on eBay from about £40 up. Top quality organ keyboards, such as from UHT are rather more expensive.

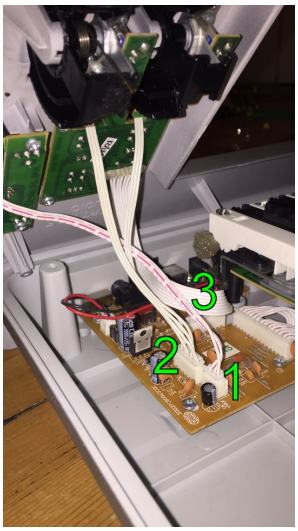
Organ pedalboards do occasionally come up for re-sale, either through organ dealers or on eBay, but a full 32 note board is rare and often expensive, even second hand. Constructing one is usually cheaper, if slower, and has the advantage of allowing the wood and design to match with the rest of the case-work.

Keyboard modification

Only the note rack and key switches are required from the 61es, not any of the MIDI electronics or the case.

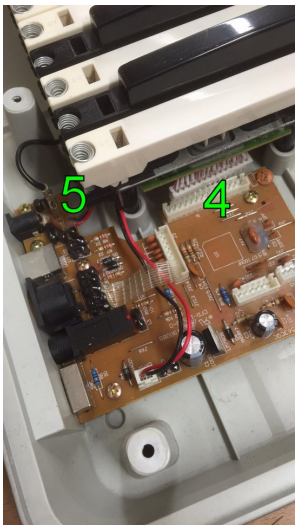


Start with the keyboard notes down, and remove all the screws around the perimeter, both the six easily seen ones along the front, and the ten in the deep rectangular holes. These are the ones marked in green in the photo. Leave the twelve in the inner ring for now. Hold the case together and turn it back over.

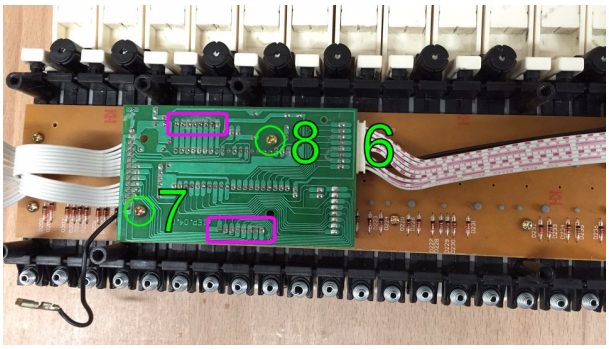


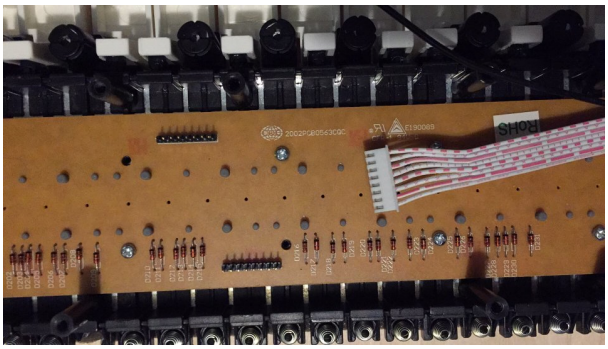
Open the case by lifting the front, and disconnect the three connectors (1 - 3) going to the top circuit board from the lower board. They just pull off vertically. The top part of the case is now removable.

Remove the long connector block and also the black grounding wire (4 & 5). Again these are just push-fit connectors, although the long connector may require careful rocking. It is not needed, so the ribbon cable can be cut through as a last resort.

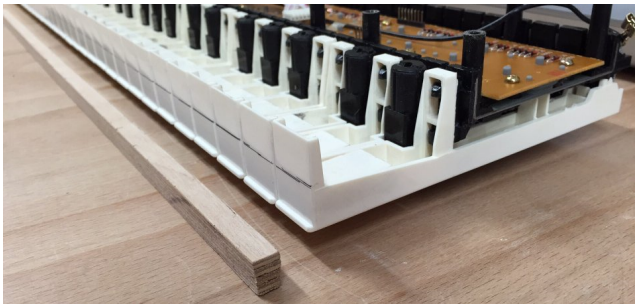


Turn the keyboard face down again and remove the twelve remaining screws. These are the ones ringed in pink in the earlier photo. The lower case half is now removable. If the keyboard is second hand, clean out the dust, fluff, spilled coffee, child inserted objects and whatever general debris has found its way inside. Musicians can be such grubby things.

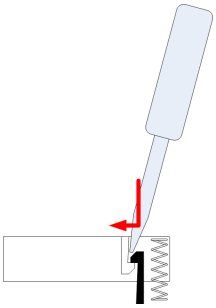




Remove the connector (6) from the side of the revealed green circuit board and undo the two screws (7 & 8) holding the board. Do not cut through this (6) ribbon cable, as it is needed. Pull this green board straight up away from the brown note circuit board. There are two on-board socket connectors, marked in pink, which mate directly with pin connectors on the note switch board. Take care not to bend these while removing the green board. There are also two, now loose, spacers that the just removed screws went through.



If under-manual preset switches are to be used, then the hanging key fronts will get in the way. Mark across the fronts where they are to be initially roughly cut. An offcut of 12mm plywood resting on the underside of the key lips is useful for this.



It is possible to remove the notes from the keyboard by removing the springs, then inserting a small screwdriver into the rectangular hole on top and pushing the note clip away from the base clip while lifting the back of the note. However, this does risk breaking one or both clips, and it is less risky, although more awkward, to trim the notes in place.

Use a rotary tool (e.g. Dremel) with an abrasive disc to cut through the keys. **Wear eye protection!** The abrasive disc is brittle and snaps easily, flying off at very high speed. One of these pieces hitting your eye will really spoil your day, and

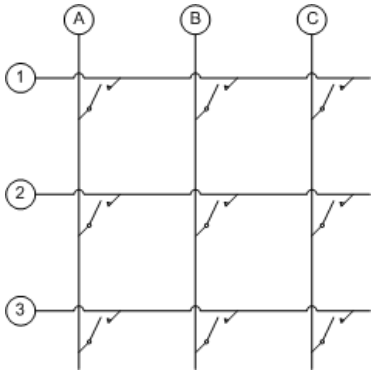
possibly your long-term sight as well. This tool will generate considerable heat and melt through the plastic as much as cut it, so do not cut closer than 1mm to where the final edge is required. The initial cut will leave a very rough edge.

Use a file to carefully smooth the cut edges. This is a long job, even with a microplane rasp which is much more efficient than a traditional file. The filing produces a lot of plastic dust and shreds, and static will stick these to whatever part of the note mechanisms they land on. If they land on any greased part of the mechanism they can be almost impossible to remove again. Before filing, drape a damp, not too wet, tissue or cloth over the surrounding plastic areas. This will help reduce the static building up and also catch some of the dust.

First technical challenge: Keyboard connection

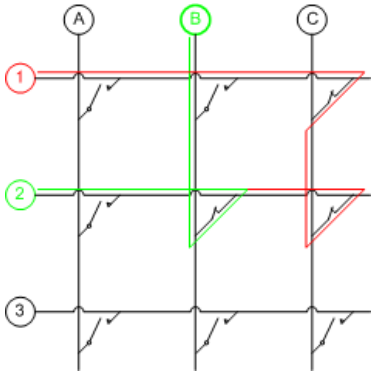
The 61es is a velocity sensitive keyboard, and it measures the key velocity by having two switches under each key, triggered at different points of the key descent. The time interval between the switches triggering indicates the velocity at which the note was pressed. A sixty one note keyboard with two switches per note means 122 switches, but there are only

twenty four connections: the two rows of eight pins and the eight wire ribbon connector.



Obviously it is not going to be possibly to independently monitor the status of each note.

Tracing the circuitry on the underside of the long brown boards shows that this is a 16x8 switch matrix. The ribbon cable and the 8 pin connector nearest the note overhang provide the 16-long side of the matrix (A-P) and the other 8 pin connector provides the 8-long side (1-8). To read which keys are pressed, power is supplied to each column A-P in turn, and the state of each row 1-8 is read.

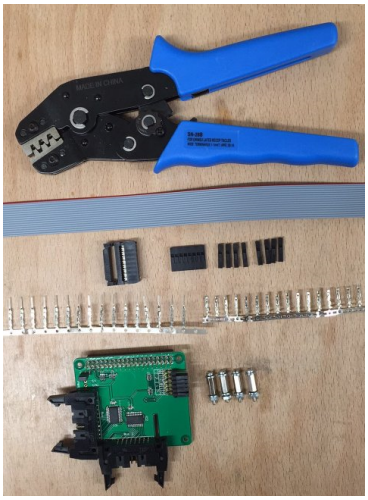


There are also diodes, not shown in the diagram here, which prevent 'ghosting' where pressing three of the four corners of any square in the matrix provides a false signal at the fourth corner. e.g. If switches B2, C1 and C2 are closed, then B2 would register when power is applied to column B, but so would B1. The diodes only allow current to flow from the column to the row, so column B to row 2 is possible, but row 2 back to column C is blocked, preventing the ghosting.

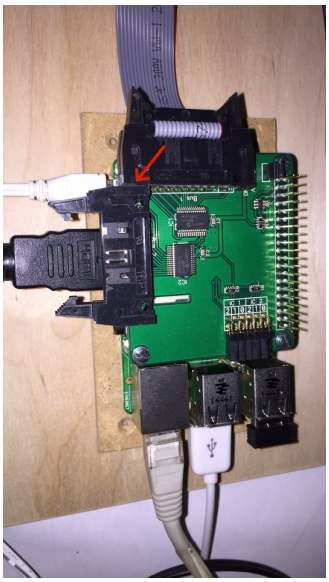
Consecutive pairs (A+B, C+D, etc) provide the two switches under each note, so only one of each needs to be used. Either the first or second of each pair can be used, but for the sake of uniform key behaviour it should be consistent across all pairs. This reduces the connectivity to a 8x8 matrix, requiring 8 outputs to provide a signal and 8 inputs to determine the keyboard state. Conveniently this is within the capability of a single MPC23017 I/O expander.

RPi I/O board

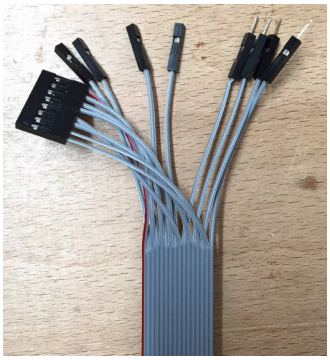
It is possible to drive a MPC23017 I/O expander directly from the RPi I^2C pins, but it uses 5V logic levels and the RPi uses 3.3V, which is below the stated threshold level of $0.8V_{DD}$ (i.e. 4V), so success is not guaranteed. Rather than using additional logic voltage level converters (e.g. these from HobbyTronics) it is more convenient to use an I/O expander board with everything already integrated, such as the IO Pi Plus board from AB Electronics,



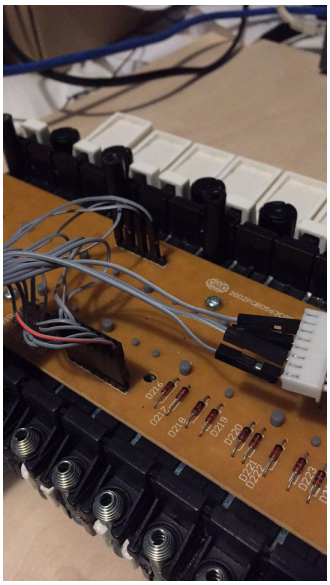
although the cost is a little higher. A couple of right angle IDC sockets make cable attachment more convenient, although the back corner of one does need filing or sanding back slightly where indicated, as they clash slightly when two are fitted. HobbyTronics also supply various crimp pins, sockets, housings and tools which are useful for making up the connecting leads.



Because of the way the pin numbers run across the IO Pi Plus connection, 1-8 in one row and 9-16 in the other, but the IDC connection alternates between rows, a very strange cable wiring is required. Starting with wire 1 (red) the switch matrix row connector needs an 8-way socket on wires 1, 3, 5, 7, 9, 11, 13 and 15; single socket connectors are required on wires 2, 4, 6 and 8; and single pin connectors are needed on 10, 12, 14 and 16. In order to help losing track of what connectors need to be crimped on where, start with putting a male (pin) crimp connector on wire 16, farthest from the red wire, then three more on alternating wires (14, 12 and 10). Then add female (socket) crimp connectors onto the remaining wires.



These then connect to the three connectors on the keyboard, with the red wire on the pin nearest the ribbon cable connector, and only alternate pins used on the other connections. These alternate connections could be placed in an 8-way housing to provide a more convenient way of handling them.



A simpler cable would have been possible, with consecutive groups of 8, 4 and 4 wires going to each connection in turn, but only at the cost of making the software scanning dependent on

using individual pins instead of the whole port. Since the intention is to keep the key-press to sound generation latency as low as possible, the more efficient code is valued over the simpler wiring. The target is to keep the delay between the note being pressed and the sound starting to under 50ms, which is typical for a tracker action wind organ.

Technical issue

In practice, the voltage drop across the diodes and the high resistance of each note switch means that feeding a logical high output from the MPC23017 into a column of the switch matrix and looking for a logical high on the rows does not work. The output voltage coming through the matrix is under 1V, which is well under the logic high voltage threshold. One workaround is to invert all signals: activate the pull-up resistors on the MPC23017 inputs and set all outputs to high except each one in turn. Pressing a note will then connect that row input back to logic low, sending the signal low for an active key press. Inverting the input signals brings the logic levels back to 0 = note off, 1 = note on, which is easier to understand when developing the software further. This is easier than adding a hardware amplification stage to the outputs to bring them back above 4V.

Testing the keyboard connectivity

A couple of the helper libraries from AB Electronics are useful. Download the repository from the [AB Electronics Github repository] and extract IoPi.py from the IOPi directory. Use the following test program to verify that all notes register, and in the correct order.

```
#!/usr/bin/python

from ABE_helpers import ABEHelpers
from ABE_IoPi import IoPi
import time
import sys

KEYADDR = 0x20

WPORT = 0
RPORT = 1

def tobin(x, count=8):
  """
  Integer to binary
  Count is number of bits
  """
  return "".join(map(lambda y:str((x>>y)&1),
range(count-1, -1, -1)))

i2c_helper = ABEHelpers()
```

```
  while True:
    # Cycle through columns
    oldstate = state
    state = ''
    time.sleep(0.005) # Wait 5ms to help
synchronise chords and lower cpu usage
    for c in range (1,9):
      bus.write_port(WPORT,0xFF ^ (1<<(c-1)))
      # Read rows
      p = bus.read_port(RPORT)
      state = state + str(tobin(p))
    change = ""
    foundchange = False
    for i in range(0,len(state)):
      if state[i] == oldstate[i]:
        change = change + "."
      else:
        change = change + state[i]
        keyseen[i] = 1
        foundchange = True
    if foundchange: print change
```

```
i2c_bus = i2c_helper.get_smbus()

bus = IoPi(i2c_bus, KEYADDR)

# Set input and output pins
# Set pull-up resistors off for write port, on
for read port
# Invert read pins
# output = 0, input = 1
bus.set_port_direction(WPORT,0x00)
bus.set_port_pullups(WPORT,0x00)
bus.set_port_direction(RPORT,0xFF)
bus.set_port_pullups(RPORT,0xFF)
bus.invert_port(RPORT,0xFF)
try:
  state = '0' * 64
  keyseen = [0] * len(state)
  foundchange = False
```

```
except KeyboardInterrupt:
  print "Cleaning up"
  bus.write_port(WPORT,0)
  print "Seen:"
  state=''
  keycount=0
  for k in keyseen:
    if k == 0:
      state = state + '0'
    else:
      state = state + '1'
      keycount = keycount + 1
  print state
  print keycount, "keys"
  sys.exit(0)
```

Test_Keyboard.py

A status line is displayed each time a change is seen, with a 1 indicating the note at that position has just been pressed, and a 0 indicating release. When interrupted (Ctrl-C) the number of different notes seen and their position is given. If playing a chromatic run through the keyboard does not trigger the note indicators in the correct sequence then check the order of the connections.

It is now a trivial matter to incorporate pyfluidsynth into this code to produce sounds based on which note is pressed, but there is no way yet of selecting which organ stops are active, or of coupling between keyboards, where pressing a note on one keyboard (or the pedals) produces a sound from the pipe set active on another keyboard.

In order to support coupling it was decided to monitor the hardware and produce the sounds in two separate programs, with the hardware monitor sending the organ control (notes and stops) status changes over the network. This permits copying the note information to a second or further system when couplers are active. In order to keep latency down, this is best done over a wired network, rather than wireless. Initially simple UDP socket transfers were used, but then this was replaced with a publish/subscribe model using the mosquitto MQTT broker. This allowed incorporation of a record/playback function which was easier to integrate with the MQTT model than with direct point-to-point network transfers. Development of the playback function showed the need, purely for cosmetic reasons, to split the visual indicators on the organ (the stops) away from the hardware

monitor, so that playback of a recorded sequence could trigger stop visualisation changes as well as changing the sounds produced.

MIDI is not a suitable format for coupling the two programs together. As well as being potentially too slow to handle large chords well, there would be some difficulty in correctly routing the signals between RPis depending on which couplers were active, although there are some ways of supporting MIDI across a network.

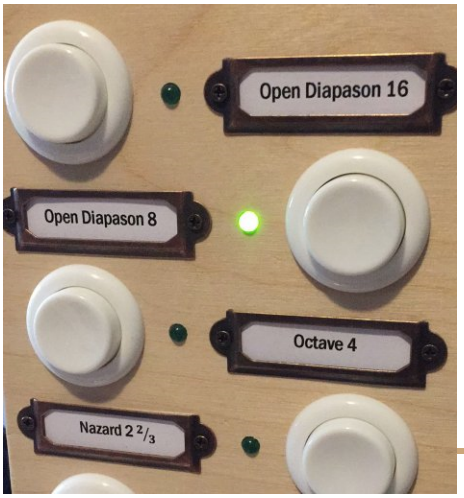
Organ Stops

In a traditional wind organ, the different sounds are produced by sending air through different pipes. The internal shaping of the mouthpiece of the pipe, and to a lesser extent, the shape of the main pipe length, produces different sounds, and then a single design is scaled suitably to produce a 'rank' of different pitches. Hence 61 pipes are need to produce enough notes for a single sound type on one keyboard, and more if there are super-octave or sub-octave couplers. The Encyclopedia of Organ Stops has internal diagrams for some of these. Airflow to each set of pipes is controlled by pulling out the stop for that set, so 'pulling out all the stops' activates the entire organ.

Apparently, very early organs had all pipe ranks active all the time, so when a mechanism was introduced to allow greater control the new ability was to allow disabling rather than enabling ranks. Hence the name 'stop' rather than 'start'. Apparently.

Second technical challenge: Stop costs

Church style organs normally use 'draw-stops' where a knob on the end of the activating rod is pulled and pushed, and cinema/theatre organs tend to use 'tab-stops' which are more like toggle switches. In both cases, the use of preset switches to allow the organist to quickly swap between combinations without having to operate possibly dozens of stops means that the stops need to be controllable (electrically or pneumatically) by the organ as well as by the organist. A modern magnetically controllable draw-stop can be about £50 once all components, and engraving the name, are included, so

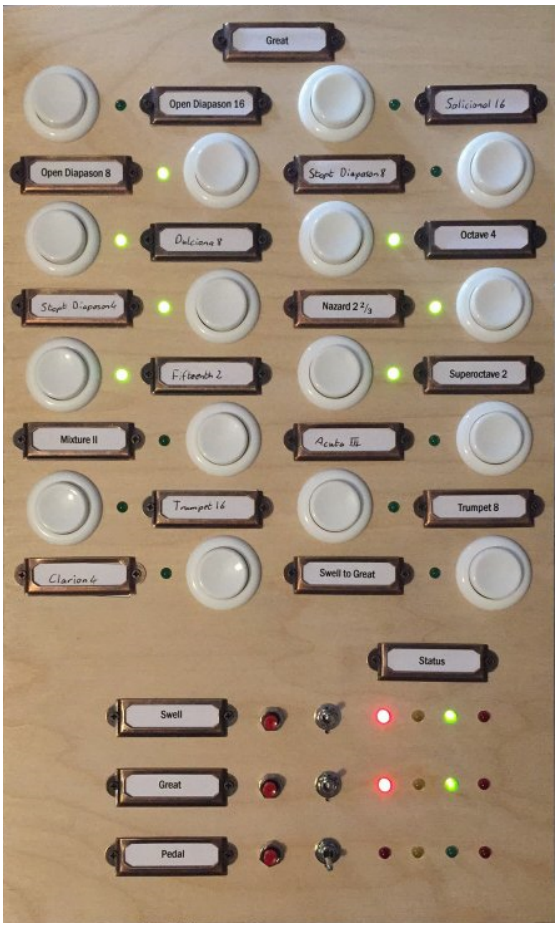


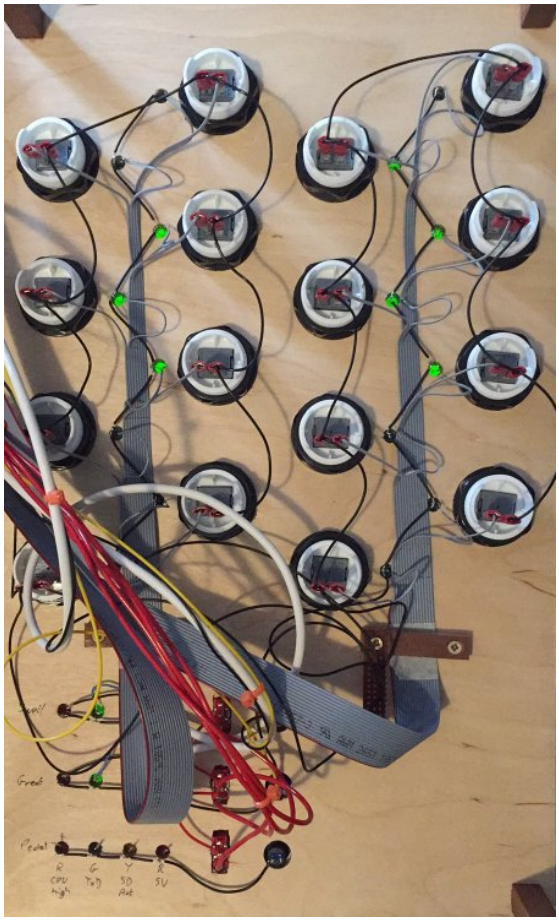
the cost of twenty or more of them is prohibitive. Non-controllable ones are simple enough to make, especially with access to a lathe, but preclude the use of presets. This project uses simple arcade-game style buttons with a nearby LED to show whether they are active or not. The buttons themselves just act as state-change triggers, and do not latch between on and off positions. In the normal nomenclature of switch types they are SPST Off-(On), meaning they only have a single pair of connections, are stable in the off position, and only stay on while pressed, then return to the off position. Only the LED indicates the status, so providing the possibility of internally switching to new stop presets. An alternative would have been a switch where the LED is internal, lighting the button itself. The stop name could even be added as an internal custom label with suitable switches. Unfortunately, this style of button normally triggers an internal microswitch, which produces a click when triggered. Leaf switches, as used here, are much quieter. Another drawback is that the designers of illuminated arcade buttons normally consider that 'brighter is better' and the current drawn by the LEDs may be too high to directly source from or sink into the IO board when more than a few are active, unless an additional resistor is added. The IO Pi Plus board used here is limited to 25mA per output, with a total of no more than 125mA per MPC23017. Driving eight LEDS from each MPC23017 chip, this means the LEDS should use no more than 15mA each. Many LEDs use 20mA, and bright ones can draw considerably more. These are suitable and the built in resistor means they can be connected across 5V with no additional components. The range includes green, yellow and red, but unfortunately not blue.

Although the required electronics to allow brighter LEDs to be used is simple (a transistor and a couple of resistors per LED, or even simplify these transistors into a couple of Darlington array ICs) this is would be another circuit board to find somewhere to mount within the organ body, and yet another tangle of internal wires. Alternatively, just a single resistor for each LED would allow current-limiting what would otherwise be an excessively bright one. Being able to wire directly to the MPC23017 inputs and outputs does simplify the wiring loom, fortunately.

All the LEDs and switches for a single manual or pedal board share a common ground, with the switch monitor lines on the IO Pi Plus being internally pulled high. This means that a triggered switch goes low when active, so like for the notes, the pin values are inverted before they are read to provide a more logically understandable value.

The interaction of the IDC wiring order and the IO Pi Plus pin layout determines the connectivity to the controls again. Here the result is that consecutive pairs of wires going to a paired switch and LED mean that switch inputs are conveniently on one port and LED outputs are on the other port.





A single MPC23017 can handle up to eight stop or coupler switches, with LEDs. Allowing for Swell to Great and Great to Swell couplers, that leaves room for seven stops on the great and swell manuals. Adding an additional IO Pi Plus to the Great manual allows eight more stops off one MPC23017 and up to sixteen preset or other control switches on the other, without having to use any more GPIO pins. However, an external power supply is required if

more than a single IO board is used. The wiring for the preset switches, mounted at the front, under the keys, has to be kept tidied out of the way of interfering with the notes. Do not forget to connect the ground side of the switches too. (The ground connection is missing in this photo!) The switches must also be mounted far enough forward that the underside of the notes does not collide with the switch body. As with the stop switches, the presets are normally pulled high and go active low when connected to ground, with the signal then being inverted.

Two IO boards are also needed for the pedalboard, as the thirty two notes are individually switched rather than being in a matrix, so require a whole IO board. The pedalboard stops hence need to be on a second IO board, which after allowing for Great to Pedal and Swell to Pedal couplers, could support fourteen more stops. Only six to eight will be provided initially, depending on what couplers are added, keeping one MPC23017 in reserve for foot-operated switches, providing suitably priced ones can be found or made.

The status panel, with only Great and Swell manuals fully wired here, shows the status of the internal Raspberries. The first red LED shows that it is powered (pin 2), the second red shows SD card activity (redirected to GPIO 23 on pin 16), green shows it is running (UART TxD on pin 8). The yellow LED is triggered when a monitoring program detects any CPU core using over 80%. This warns the organist not to pull out any more stops, as if the synthesizer cannot cope with the load then the output becomes extremely distorted. The push-button switch, when held for at least three seconds, triggers a safe shutdown of the Pi. The toggle switch is for the Pi power lead, allowing it to be restarted after a shutdown, or to force it to stay off when the organ mains power is disconnected and reconnected.

Great manual	IO card 1	IO1: Notes 1-61 IO2: Stops 1-8
	IO card 2	IO1: Presets 1-16 IO2: Stops 9-15, Coupler 1
	GPIO	Shutdown switch SD card activity High CPU warning LED
Swell manual	IO card 1	IO1: Notes 1-61 IO2: Stops 1-7, Coupler 1

	IO card 2	IO1: Presets 1-16 IO2: Unused
	GPIO	Shutdown switch SD card activity High CPU warning LED
Pedalboard	IO card 1	IO1: Notes 1-16 IO2: Notes 17-32
	IO card 2	IO1: Presets, if used IO2: Stops 1-6, Couplers 1-2
	GPIO	Shutdown switch SD card activity High CPU warning LED

Latency and polyphony

The goal is to have the delay between pressing the key and the note sounding to be in the region of 50 milliseconds, to be comparable with a small church organ. In a very large organ the pipes may be sited some distance from the organ console, so an additional delay is introduced before the organist hears the results of what they are playing, of roughly 3ms per meter.

The tracker rods of a wind organ should activate the valves in the wind chest when the note key is half way through its travel. By holding a microphone close to the keys and tapping them it is possible to record the sound of the finger hitting the key, the key hitting the key bed and the note playing. Pressing the note slowly and listening for the note onset reveals the activation point of the note. In this case it is about two thirds of the way through the travel. Examining the recording in an audio editor can then show the total latency between two thirds of the way though the key travel and the start of the note.

The slightly late note activation position can be considered an advantage, depending on the organist's (lack of) skill. An organ with a light touch and an early activation point is very easy to trigger unwanted notes on, just by gently brushing a note neighbouring the correct one.

The critical element of this system, as far as sound quality is concerned, is the synthesizer. In order to provide it with the maximum resource level possible, the term `isolcpus=3` is added to /boot/cmdline.txt. This prevents any process running on CPU core 3 unless explicitly set there. The sound generator can then be started with `taskset -c 3` to restrict it to this core. It now has exclusive use of one CPU core. Since it is not multi-threaded this is now providing it as much dedicated processing power as possible. Multi-core support is an experimental level feature of FluidSynth, but currently the overhead of preparing the parallelization balances out any gain in performance when being used in a low-latency situation.

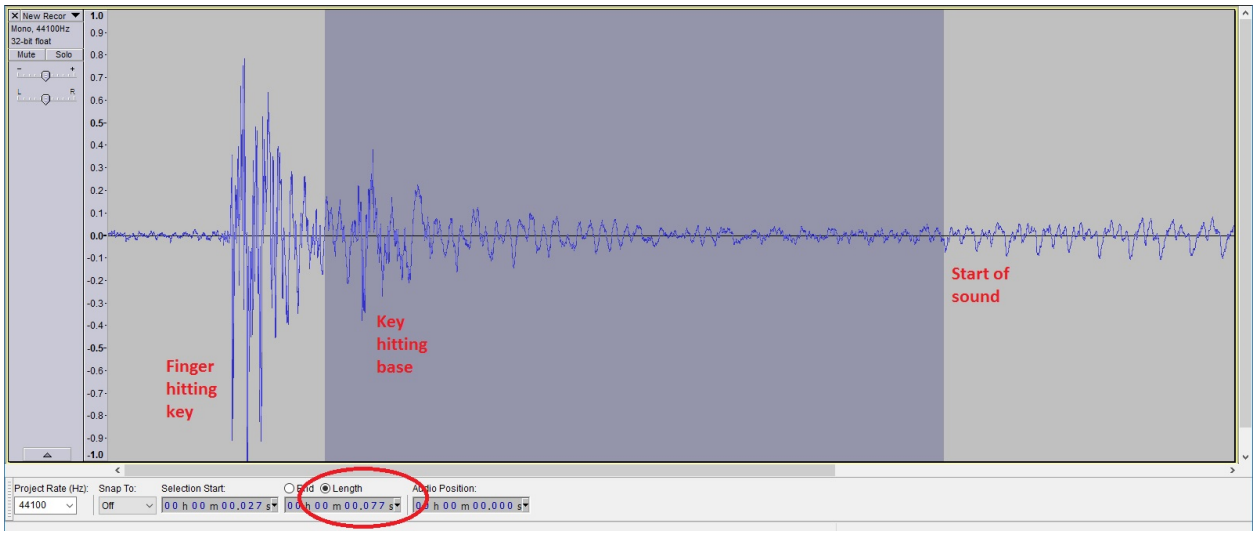
A further reduction in CPU usage could be achieved by using a better soundcard. An I^2S based DAC card such as from HiFiBerry or IQaudIO does not use the USB transfer mechanism, so reduces CPU usage and also does not risk interruption from having the USB and network sharing a common platform. However, the use of these cards is not recommended on a RPi with other I/O cards, although it should be possible, provided care is taken to avoid clashing GPIO ports and I^2C bus usage. This may become a future enhancement. An alternative is to have the hardware interaction controlled by one RPi and the sound generation performed by another, so moving the soundcard away from the I/O boards. Six RPis in a project such as this might be considered excessive, and space and power issues need to be considered. Although the total power consumption would still be reasonably low (probably lower than the audio amplifier draws), it is all at 5V and the current would exceed 10A.

A soundfont works by taking one or more sound samples, potentially performing assorted transformations on them, then combining them. The Jeux soundfont turned out to use a lot of transformations on generally two or three samples per stop, producing a great sound but at a high computational cost. A Raspberry Pi 2B was limited to about 25 note polyphony using this soundfont, so a seven note chord with four stops active was likely to distort as the processor could not calculate the audio buffer contents fast enough. With the more complex stop sounds, the limit could be even lower. The solution to this was two-fold: switch to the English Organ soundfont, which uses single samples with no additional processing and use the increased power of the Raspberry Pi 3B instead, which had just become available at this stage in the development. This gives a polyphony limit of

approximately 170. With only fifteen stops available on the Great manual, the organist should run out of fingers before hitting the limit. However, the processing load seems to be higher at the start and end of the notes, so it is still a good idea to stick with no more than eleven stops at a time to avoid occasional distortion on large chords.

The two programs, handling the hardware monitoring and the sound generation, can be instrumented to report the average cycle time for a single full hardware scan, and for turning that notification into a command to the synthesizer to produce a sound. When the two programs are running on a common system, the delay between the signal send from the first and arriving at the second is also easily found.

The timings are approximately 5-6ms to scan the hardware, including a 3ms delay to help synchronise chord capture (it is better, on the scale of a few milliseconds, to wait and see if any more notes are to be pressed or released and send them all together than to send multiple network packets containing one note each), and to reduce the CPU load; 1-2ms for interprocess communication, which is likely to be nearer 3-4ms when coupling across to a different keyboard over the network; and about 25ms between receiving the command and relaying it to the synthesizer. There will then be an additional delay for the synthesizer to populate the next sound buffer, then transfer it through the audio output.



Recording a key being tapped shows the delay to the start of the sound being 77ms from two thirds of the way through the key travel, so this implies about 45ms delay within the synthesizer. This is too long, so some optimisation is required.

Tuning of the sound buffer parameters used by FluidSynth is made easier by applying the following patch to /usr/local/lib/python2.7/dist-packages/fluidsynth.py.

```
181c181
<     def __init__(self, gain=0.2, samplerate=44100):
---
>     def __init__(self, gain=0.2, samplerate=44100, polyphony=256, midichannels=16,
periods=16, periodsize=64, reverb='yes', chorus='yes'):
193,194c193,198
<         # No reason to limit ourselves to 16 channels
<         fluid_settings_setint(st, 'synth.midi-channels', 256)
---
>         fluid_settings_setint(st, 'synth.polyphony', polyphony)
>         fluid_settings_setint(st, 'synth.midi-channels', midichannels)
>         fluid_settings_setint(st, 'synth.periods', periods)
>         fluid_settings_setint(st, 'synth.period-size', periodsize)
>         fluid_settings_setstr(st, 'synth.reverb.active', reverb)
>         fluid_settings_setstr(st, 'synth.chorus.active', chorus)
```

The default values given here are the default FluidSynth values for a Linux platform. Reducing the periods to 2 and disabling reverb and chorus should the latency and lower the CPU usage. Reverberation is then added back by an external effects unit in the audio mixing desk, which does a better job. Reducing the number of periods and the period size reduces the latency, but reducing them too far runs the risk of a kernel interrupt preventing processing being carried out within the available time window, leading to distortion again. Compiling a low-latency kernel can help with this, but this is a major step away from being able to keep the system up-to-date through the `apt-get upgrade` mechanism.

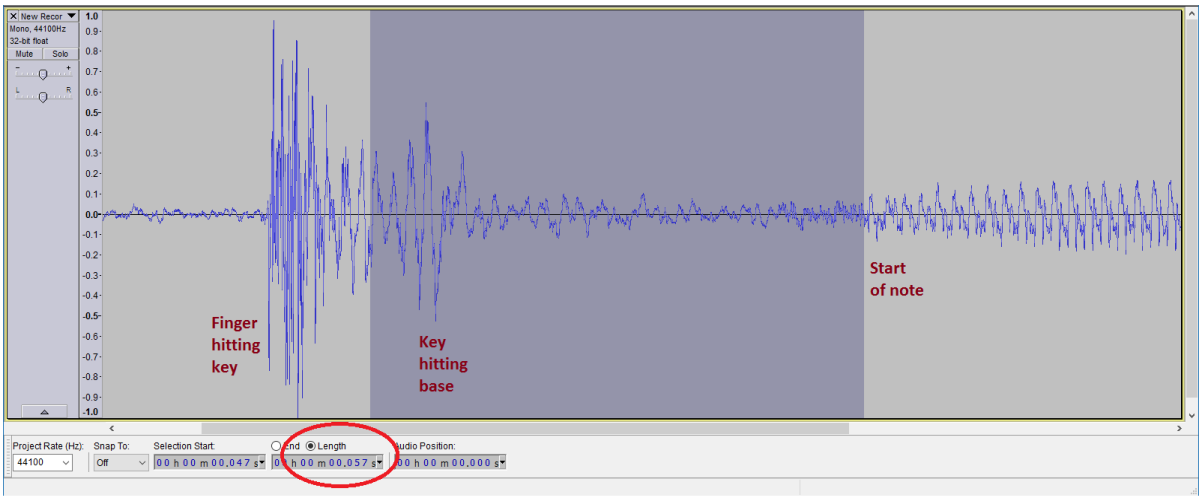
Reducing the polyphony is also reported to reduce the CPU requirements, but an organ manual is capable of producing a lot of sounds simultaneously (10 fingers x 15 stops = 150 notes). A trial with a polyphony limit of 32 showed that it was very noticeable that notes were being dropped out of large chords.

Various methods of storing and updating the stop and note status within the sound generation half of the software were also tried and compared for speed. Surprisingly, using numpy, which is intended for numerical processing and might be considered to thus have been tuned for speed, proved to be considerably slower when using 2D arrays than using standard Python lists of lists. The time taken to process state changes is proportional to the number of stops, so the Swell and Pedal organs will require less processing than the Great

organ. Changing the data structure cut the average processing time from 25ms to 4ms for the 15-stop Great organ.

A trial of Python v3.4 against v2.7 showed that v3 was considerably slower, so all code is written in Python 2. A useful benchmark of the speeds of accessing GPIO ports available at http://codeandlife.com shows that they can be accessed much faster through C than through Python, but the hardware interface is not the limiting factor here.

After these optimisations, the sound delay dropped from 77ms to 57ms. This can be accounted for entirely by the note/stop processing part of the sound generation latency dropping from 25ms to 4ms, so it would appear the change in FluidSynth settings are having no noticeable effect. In addition, the loudspeakers were about 2m further away from the recording microphone than the note being pressed, so the real internal latency is about 6ms shorter.



Debugging

Two levels of console output are allowed for by command line options for both the hardware monitor and sound synthesis parts of the code. A ‘verbose’ option allows primarily initialisation details to be displayed, allowing verification of the configuration file options being applied, whereas a ‘debug’ option displays every change detected in hardware status, transmission and reception of network data, changes to the synthesizer state machine and notification of sound generation.

Once all easy-to-find bugs have been found, take a device with nearly 200 assorted switches (notes, stops, presets, etc), most of which play a sound or change a light, and then let a four year old loose on it. Watch for unconventional organ behaviour resulting from unconventional organist behaviour, then fix those bugs too.

Appendix A - Hardware providers

CPC: Just about anything electrical and a fair amount mechanical. http://www.cpc.co.uk

Hobbytronics: Other useful electronics when the choice is overwhelming at CPC. http://www.hobbytronics.co.uk

AB Electronics UK: RPi interface boards. https://www.abelectronics.co.uk/

Arcade World UK: Buttons are a lot cheaper than proper stops. http://www.arcadeworlduk.com/

eBay: When all else fails. http://www.ebay.co.uk/

Appendix B - Software / data / references

English Organ soundfont: http://www.milestones.me.uk/soundfonts.html

VirtuOrgan soundfont: https://musescore.org/en/node/47886

Jeux soundfont: http://www.realmac.info/jeux1.htm

Other soundfount collections: http://midiorgan.com/virtualsamples/ and https://stratmaninstruments.joomla.com/jorgan-disposition-packages

AB Electronics IO card libraries: https://github.com/abelectronicsuk/ABElectronics_Python_Libraries

Colin Pykett: http://www.pykett.org.uk/index.htm A large collection of articles on the whats and hows of organs, both traditional and VPOs.

Appendix C - Raspberry Pi Configuration

Start with Stretch-lite Raspbian image

Use raspi-config to expand file system, set the hostname, WiFi country, SSH daemon enabled, I2C interfaces, etc.

Edit /etc/network/interfaces to set any required network configuration.

Rebuild the ssh keys so that they are not the distribution standard ones:

```
sudo rm /etc/ssh/ssh_host_*
sudo dpkg-reconfigure openssh-server
```

Update to the latest version of everything

```
sudo apt update
sudo apt upgrade
```

Add or change the lines in /boot/config.txt

```
dtparam=i2c_arm=on
dtparam=audio=off
gpu_mem=16
dtparam=act_led_gpio=23 # RPi 2 only, not 3
```

/etc/apt/sources.list may need to add the line

```
deb http://mirrordirector.raspbian.org/raspbian/ stretch main contrib
non-free rpi
```

Install needed packages

```
sudo apt install python-pip python-smbus python-dev python-setuptools
sudo apt install fluidsynth
sudo apt install mosquitto mosquitto-clients
sudo pip install pyfluidsynth
sudo pip install configparser
sudo pip install psutil
sudo pip install paho_mqtt
```

Add the following into /etc/modules-load.d/modules.conf

```
i2c-dev
snd-seq-midi
```

Install eorg104a soundfont from http://www.milestones.me.uk/soundfonts.html
Sfarkxtc will be needed to unpack it, from http://melodymachine.com/sfark-linux-mac

Add the following into /etc/rc.local

```
for cpu in /sys/devices/system/cpu/cpu[0-9]*; do echo -n performance >
$cpu/cpufreq/scaling-governor; done
```

Put '`blacklist ipv6`' in /etc/modprobe.d/raspiv-blacklist.conf. Not for any audio purposes, but it is big and not needed. For the RPi3 also add '`blacklist btbcm`' and '`blacklist hci_uart`' unless Bluetooth is needed.

Comment out the `options snd-usb-audio index=-2` line in /lib/modprobe.d/aliases.conf

'Device' is the name of the usb soundcard after 'card *n*:' as shown by `aplay -l`

Put the following in /etc/asound.conf:

```
pcm.!default {
  type hw
  Card Device
}
ctl.!default {
  type hw
  card Device
}
```

Some soundcards are locked to the frequencies they support (e.g. 48KHz but not 44.1KHz). For these, either match the frequency used in the FluidSynth configuration, or go through the software conversion in the sound system:

```
pcm.!default {
  type plug
  slave {
    pcm "hw:Device,0"
  }
}
ctl.!default {
  type hw
  card Device
}
```

Add the pi user to the audio group

```
usermod -a -G audio pi
```

Put the following in /etc/security/limits.d/audio.conf

```
@audio   -  rtprio    95
@audio   -  memlock   unlimited
```

Edit/create /etc/systemd/system/networking.service.d/reduce-timeout.conf

```
[Service]
TimeoutStartSec=30
```

For the RPi3, create an overlay to allow act_led redirection.

```
// act_led_redirect.dts
// Redirect the act_led
/dts-v1/;
/plugin/;
/ {
```

```
compatible = "brcm,bcm2710";

fragment@0 {
      target = <&leds>;
      __overlay__ {
            act_led: act {
                  label = "led0";
                  linux,default-trigger = "mmc0";
                  gpios = <&gpio 23 0>;
                  };
            };
      };
};
```

Compile it

```
apt-get install device-tree-compiler
dtc -@ -I dts -O dtb act_led_redirect.dts > act_led_redirect.dtb
```

Copy the dtb into /boot/overlays and enable by adding `dtoverlay=act_led_redirect` to /boot/config.txt.

Later versions of Raspbian brought back a standard overlay that served the same purpose, so adding `dtoverlay=pi3-act-led,gpio=23` may be all that is required.

Also add `dtoverlay=pi3-disable-bt` to /boot/config.txt to re-enable the use of the GPIO UART, so lighting the status LED.

Finally, add `isolcpus=3 ipv6.disable=1` to /boot/cmdline.txt to allow a dedicate cpu core and to further block IPv6..

Reboot