S-2020 Audio System

Nick Glazzard

January 3, 2017

Contents

1 Introduction

S-2020 is a series of DIY audio components to replace the Arcam Alpha 10 DAB Tuner, Meridian G91 DVD Audio Player / Controller / Tuner, and DSP-5000 Loudspeakers that were sold in Spring 2016. The components are being built over an extended period of time, coming into service as they are completed. They are being used with a 'low end' system which they will eventually replace (perhaps). The non-S-2020 components are currently:

- Denon PMA–255UK Integrated Amplifier. This was bought in 2000, but has seen little use. It can be bought on EBay for less than £50. This is being used only as a power amplifier.
- JPW Sonata 'bookshelf' loudspeakers. These cost £50 from EBay in mint condition. They date back to the early 1990s.

In spite of these items being available for under £100 in total, it seems to my ears that they make very pleasant sounds. The loudspeakers are naturally not capable of reproducing deep bass and would probably be poor if used at very high volumes, but otherwise are very good indeed[1].

The goals for S-2020 are:

- Use technologies and methods appropriate to the third decade of the 21st Century. Hence the name.
- Be easily maintainable (and reasonably easy to construct in the first place).
- Produce excellent results (of course!).

So far, two components have been completed:

- S-2020-S Universal Audio Source.
- S-2020-P Pre-amplifier (or, perhaps better, controller, since it does no actual amplification).

These are described fully in this document.

Possible future components are:

- S-2020-A Power Amplifier
- DIY loudspeakers of some kind.

Whether these will actually be built is not certain. This is for a number of reasons:

- The loudspeakers I would like to try are Siegfried Linkwitz's LXmini design (see: http://www.linkwitzlab.com). However, it is highly unlikely that the appearance of these would be acceptable domestically.

[1] To my ears, in my room, etc. Your mileage may vary . . . Feel free to conclude that I must be deaf.

- The Denon PMA-255UK gives a pretty unimpeachable performance[2]
- The JPW loudspeakers are sufficiently good (to my ears) to make me wonder if any alternative, traditionally designed, loudspeaker would *actually* be better — *all things considered* (and I don't mean purchase price by that) [3].

Using the S-2020 system (at least, the S-2020-S component) does require some technical skill. It might be concluded that this is a system for 'geeks' — and that is probably true to some extent. As described here, it does assume some computer infrastructure in the home — something that runs a web browser (a tablet, ideally), an Internet router, a WiFi network, an Ubuntu Linux computing environment for ripping CDs and uploading music[4], willingness to use a command line interface[5] . . . Perhaps not everyone's cup of tea.

On the other hand, pretty much any kind of what might be called 'file based audio' is going to involve technical know how at some level (above zero!) no matter what specific platform(s) may be involved. This is just part of life as we approach the 21st Century's third decade.

It would be possible to hide some of the details laid out in the documentation. Presumably, at least some 'music server' systems do a good job at hiding all this sort of thing (which might justify their expense for many people), but the 'geeky' machinery will still be there behind the scenes. At least spelling out what is really going on gives people a chance to find fixes and workarounds to problems that will almost certainly arise — if not today, then at some point in the face of constant updates[6] to every type of modern computing infrastructure.

[2]This is as it should be: all really competently designed and constructed amplifiers should sound the same — and for all practical purposes they should be perfect 'wires with gain' (up to their design limits). Although designing and building such an amplifier from scratch is not a simple task even today, there are components which *do* make it easy — e.g. LM3886 devices or Class D amplifier modules, such as those from Hypex. If two channels at modest power is what is required, though, it would be difficult to do better than the Denon. The LXmini design requires four channels, so that would be a reason to build the S-2020-A component.

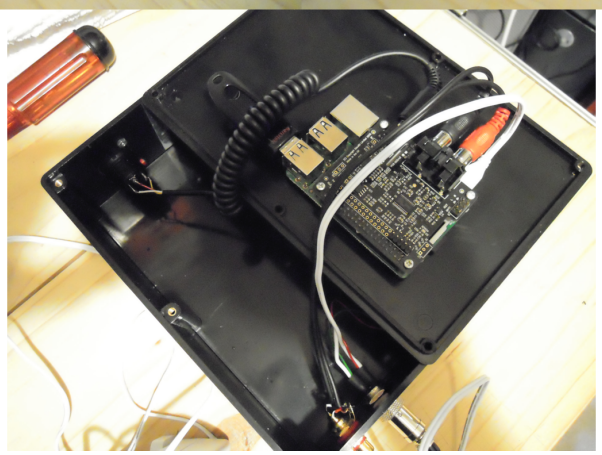
[3]This would seem to be a *ridiculous conclusion* — certainly for anyone who believes that the most important part of a component's specification is its price tag or brand name. *However* . . . **(private comments redacted)**. It really does seem that the only way to select loudspeakers is to listen to them at some length — which for most people, me included, means *there is no good, practical, way to choose a loudspeaker* — and certainly no way *to be sure* that any given model will be better (or worse) than what you have. Price is *definitely not* a useful guide.

[4]There are many ways of doing this, actually — but none of them would be totally painless for technophobes.

[5]This remains the easiest and clearest way of doing some things — although very few people can be convinced of that!

[6]This makes 'file based' or 'computer based' audio systems significantly different from traditional equipment. Things they rely on change. This may be particularly true for Internet radio and streaming services.

2 The S-2020-S Universal Audio Source



This takes the place of what, in the past, would have been several different components. It provides the following:

- Storage and replay of a local, private, music collection. This is material bought as CDs, LPs, downloads and so forth. This element replaces all optical disk players, turntables and so on.
- Internet radio access. This element replaces FM and DAB tuners.
- Streaming service access. This is currently restricted to Spotify Premium. This arguably replaces the need to buy downloads, CDs and LPs in the future. I wouldn't say it does, entirely, but it could be seen that way.

The system is built around the following main hardware components:

- Raspberry Pi 3 Model B embedded computer — based on a dual core ARM processor with built-in Wi-Fi. See: https://www.raspberrypi.org/products/raspberry-pi-3-model-b.
- IQaudIO Pi-DAC+ HAT for the Raspberry Pi 3. This provides very high quality analog audio at line level and has a built in headphone amplifier. See: http://www.iqaudio.co.uk.

- SanDisk Ultra 256 GB USB 3 Flash Drive. This provides adequately large, fast, totally silent storage for the local music collection (in the form of FLAC files — no lossy compression).
- Official Raspberry Pi 2 amp power supply. This provides 5.2V on a decently thick cable. Note that many 'chargers' and power supplies are not adequate in practice, regardless of their specifications.

These major items are supplemented with:

- A plastic enclosure large enough to hold all parts. This is an Electromart 220 X 150 X 64MM ABS box. See: `http://stores.ebay.co.uk/electromartuk2000/`. Note that this *must* be plastic so that the Raspberry Pi 3's built-in Wi-Fi will work.
- A set of four 13MM X 13MM X 6MM stick-on rubber feet for the plastic enclosure, also from Electromart.
- One or two pairs of gold plated RCA phono panel mount chassis sockets. These are used with a short phono plug terminated coaxial cable pair to bring analog audio out and eliminate stress on the IQaudIO card.
- One Neutrik NJ3FP6C-BAG 1/4" stereo jack panel mount socket. This is connected to a 3.5mm jack plug terminated coaxial cable pair to bring the headphone connection out and eliminate stress on the IQaudIO card.
- A GX16-2 2 pin locking chassis panel socket and matching plug. This is used to supply power to the box. The original Raspberry Pi power supply cable is cut and this combination inserted. Note that the standard 2.5mm (or similar) power sockets are, in my opinion, not fit for purpose. I tried two and neither provided a reliable connection. This solution is vastly more robust.

The software used with this hardware is all open source — none of it written by me, I hasten to add! See Appendix G for the people to thank for it. The software items are:

- piCorePlayer V2.05
  See: `https://sites.google.com/site/picoreplayer/home`. This is a turnkey system for the Raspberry Pi 3 that acts as a replacement for a Slim Devices or Logitech Squeezebox player. It can make full use of the IQaudIO PiDAC+ I2S-DAC. As of March 2016, it can also host Logitech Media Server. It is built on TinyCore Linux.
- Logitech Media Server (LMS) V7.9
  This is a very complete audio server that I have used for many years. It is controlled via any Web browser or one of a number of applications on Android and iOS tablets and phones. It can be installed trivially on a piCorePlayer system.
- Squeezelite V1.8.4
  This is the software emulation of a Squeezebox player supplied as part of piCorePlayer.
- LMS BBC iPlayer plug-in (Triode)
  This enables access to BBC iPlayer. Other Internet radio 'stations' are available in LMS without any plug-in, but the BBC stations are a special case. Rather too special, in fact . . . more below.

- LMS Spotify plugin (unofficial) (Triode)
  This provides efficient access to the Spotify Premium streaming service. It works very well once installed.

Two S-2020-S devices were built and given the names: piCorePlayer and piCoreRed. They are almost identical, but, of course, have different Wi-Fi MAC addresses. The two can be distinguished by:

- piCorePlayer
  - MAC: `b8:27:eb:26:57:76`
  - Host name: `piCorePlayer`
  - LMS player name: `piCorePlayer`
  - Green power LED.
  - Two pairs of audio outputs (in parallel). One is used as the main output and one to go to a recording device.
  - LMS audio library folder: `/mnt/audiolib`
- piCoreRed
  - MAC: `b8:27:eb:43:fe:e9`
  - Host name: `piCoreRed`
  - LMS player name: `piCoreRed`
  - Red power LED.
  - One pair of audio outputs.
  - LMS audio library folder: `/mnt/sda1/audiolib`

Ideally, the local network's DHCP server would be set up to always assign a fixed IP address to the MAC addresses for the devices, which could then be identified by name via a hosts file (or an equivalent method). Unfortunately, my router has this facility in its DHCP server, but it doesn't actually work! Hence the IP addresses that were assigned to the S-2020-S devices must be found by an IP scanner (e.g. Net Analyzer for iOS) and used directly. They may change after power outages or other events. This is not the fault of the S-2020-S. I have now replaced the router's DHCP server with an alternative, as described in Appendix A.

The primary means of controlling the S-2020-S is via a Web browser.

A comprehensive interface to the piCorePlayer system itself can be accessed via:
`http://<IP-Address>` where `<IP-Address>` is the IP address assigned to the S-2020-S by DHCP. This presents the piCorePlayer 'main page' from which tabs can be used to access other configuration pages (see Figure 1).

Once the S-2020-S system has been configured, there is no reason to use this page — unless problems arise.

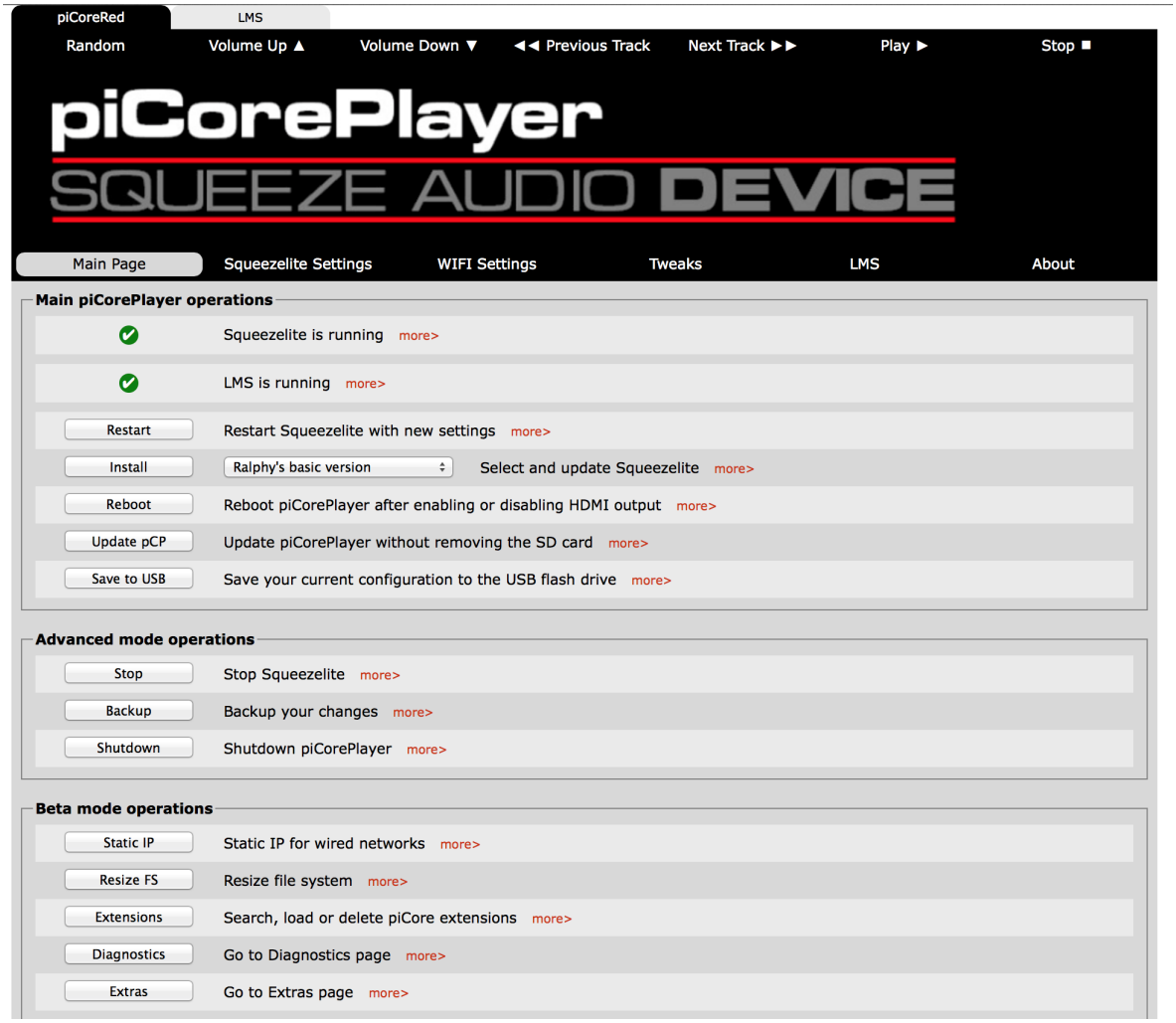


Figure 1: The piCorePlayer main page

What is played (and at what volume, in the first instance) is controlled through the LMS interface. This can be accessed at: `http://<IP-Address>:9000`
This presents a fairly self explanatory interface, an example of which is shown in Figure 2.

Pressing the **Settings** button brings up the **LMS Settings** page. The main use for this is to update the library after adding new material, or to configure Internet radio stations. These tasks are described below.

2.1 Updating the Library

Given a folder containing a set of FLAC files to be uploaded to LMS, the easiest way of doing this is with `scp` — the command line Secure Copy Program.

For **piCorePlayer**:

```
scp -r "album folder"  tc@<ip address>:/mnt/audiolib
```

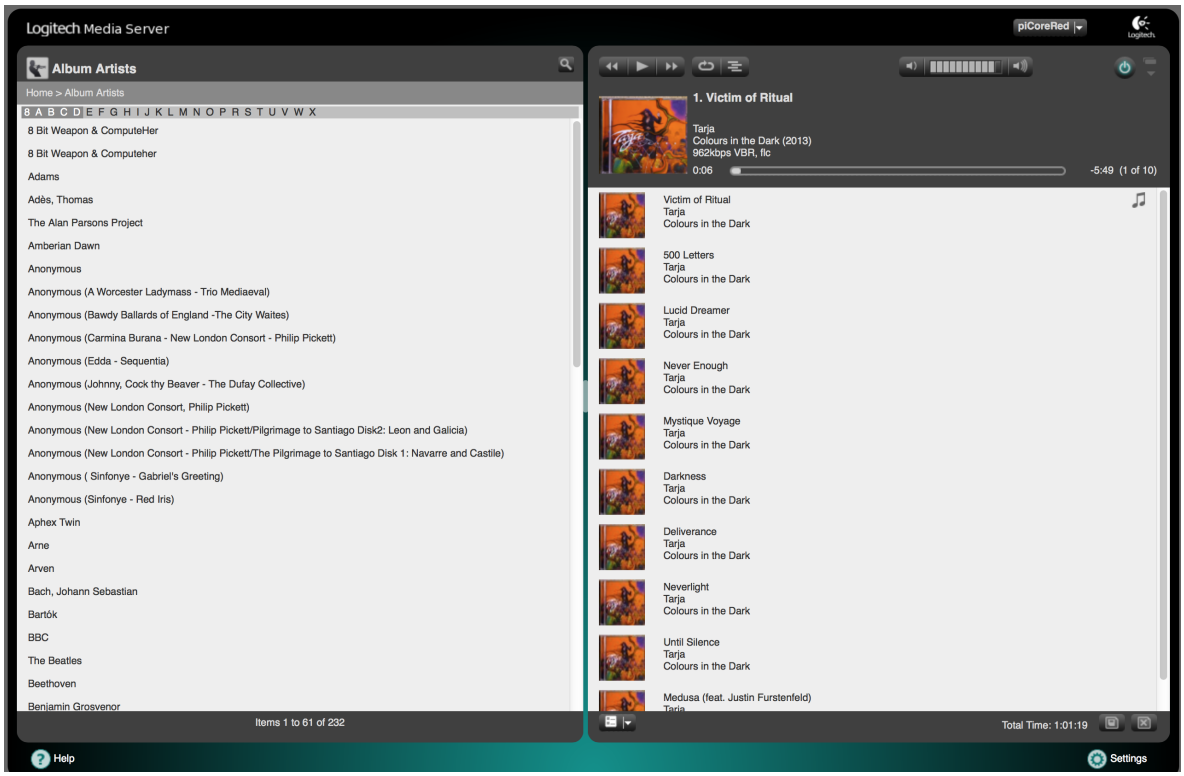


Figure 2: An LMS main page example

For **piCoreRed**:

```
scp -r "album folder"  tc@<ip address>:/mnt/sda1/audiolib
```

Ripping CDs for uploading is readily done using standard Ubuntu Linux tools. I do this on the `nick-Lenovo-T60` laptop which runs Ubuntu 14.10 LTS.

- **RIP with: Sound Juicer**
  Output to: `~/Music/artist/album`
- **Scan CD cover with: SimpleScan**
  Put the image in the `artist` directory.
- **Add the cover to the FLAC files**:

```
cd artist/album
metaflac --import-picture-from=../scan.jpg *.flac
```

- **Other potentially useful tag manipulations**:

```
metaflac --remove-tag=album *.flac
metaflac --set-tag=album=An album *.flac
metaflac --set-tag=albumartist=Album artist *.flac
metaflac --show-tag=album *.flac
```

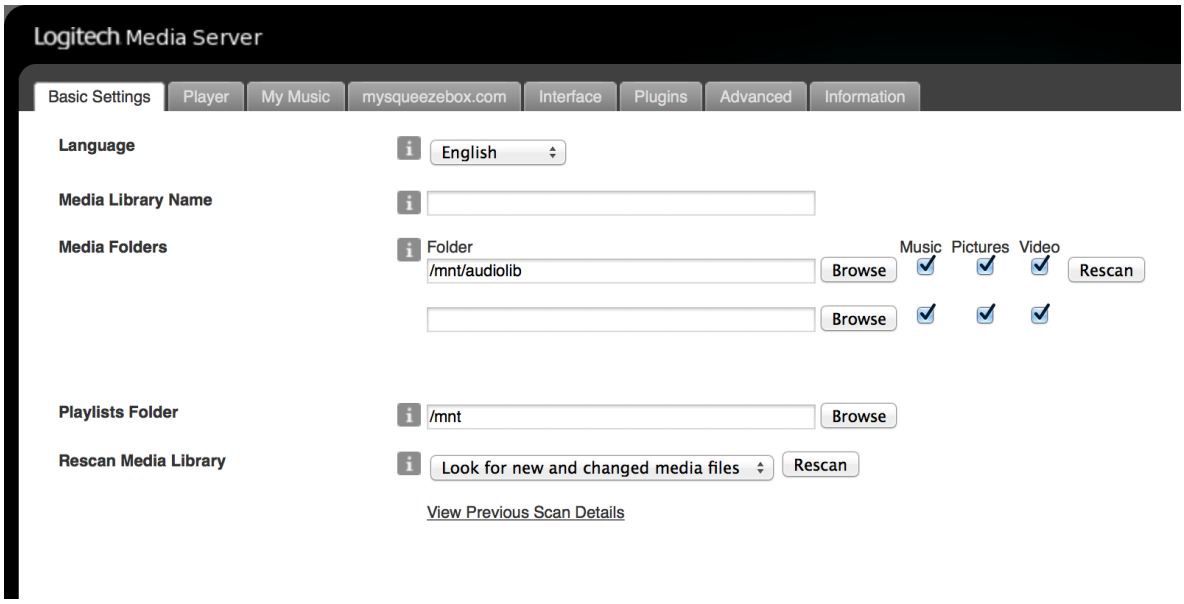


Figure 3: Instructing LMS to scan an updated library

Once the new files have been transferred to the audio library folder, LMS must be told to rescan the library. This is done by pressing the **Rescan** button in the **Basic Settings** tab of the LMS **Settings** page, as shown in Figure 3.

A shell script that greatly simplifies the updating of the audio libraries from ripped CD material is described in Appendix B. This is used on the Ubuntu Linux machine used for ripping. Some other configuration commands are also described which make things much more convenient than they otherwise would be.

2.2 Configuring an Internet Radio Station

To add a non-BBC radio station to those available and to start listening to it, simply go to:

```
Home > Tune In URL
```

on the LMS page, then enter the URL and press `Tune In`. The station will begin playing. To add it to a list of 'presets', press the Add button. As an example, see Figure 4, which is setting the URL for BR Klassik (high bit rate service). This could hardly be easier.

The quality of a service such as this example (128kb/s MP3) is better than 192kb/s DAB and is superior to FM[7].

2.3 Listening to BBC Radio Stations

It is only possible to listen to BBC radio stations using Triode's iPlayer plug-in to LMS. This is excellent software and can work very well, but there are some problems inherent to the way the BBC has chosen to do things (which is unique, as far as I know).

[7] In my opinion. Although it must be said that a really good FM tuner — such as my carefully fettled ReVoX A76 —

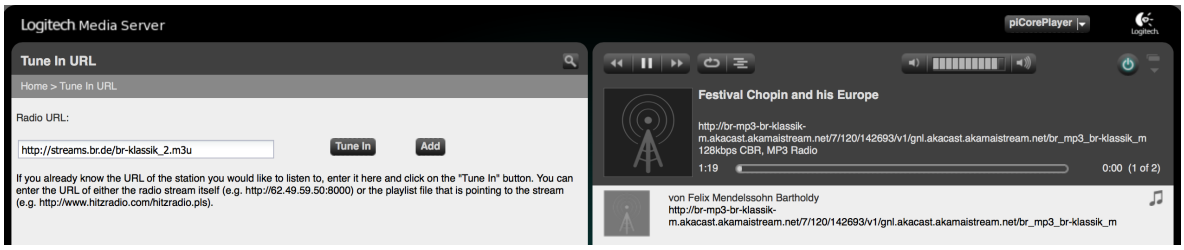


Figure 4: Tuning in a non-BBC radio station

To listen to a BBC radio station (once the iPlayer plug-in has been installed), go to:

```
Home > BBC iPlayer > Listen Live
```

and choose a station (e.g. BBC Radio 3).

In principle, the BBC iPlayer can provide the highest quality Internet radio service in the world. This is the case for Radio 3, which is broadcast as a 339kb/s AAC stream.

2.3.1 BBC Radio iPlayer Problems and Solutions

Unfortunately, the service is streamed using HLS technology (HTTP Live Streaming). This was introduced by the BBC rather suddenly in early 2015 and broke Triode's iPlayer plug-in. This was fixed quickly, but a further change to DASH (Dynamic Adaptive Streaming over HTTP) at some point in the future seems likely. The problem with HLS and DASH — at least, when they are used in the way the BBC uses them — is that they require very robust DNS lookup support. Material seems to be sent out in chunks of a few seconds, the retrieval of each of which involves a separate DNS lookup. If these time-out, the service will stop. There also seems to be some DNS related issue that causes audible glitches every few seconds — they sound a bit like scratches on a vinyl record and probably are due to very short segments of signal being missed out. It seems odd that this could be a DNS issue, but changing DNS servers fixed this for me — or made it very rare — so it must be. Google Public DNS server 8.8.4.4 is the fastest according to **namebench** at my location and using it made a *huge* improvement.

Although BBC Radio iPlayer is a very high quality service once DNS related problems have been solved, it is still significantly less reliable than some other services — e.g. BR Klassik from Munich. It just stops randomly, rarely exceeding 2 hours of uninterrupted replay, requiring it to be manually restarted (which always restores the service immediately). No doubt something network related is at the bottom of this — perhaps DNS time-outs, but I have done all I can to fix them. The service is definitely fragile.

2.4 Configuring Spotify Premium

To use the Spotify streaming service, you must have a Spotify Premium account — one that you pay for and which comes with a user name and password.

is hard to beat given a strong, clean signal from a good aerial.

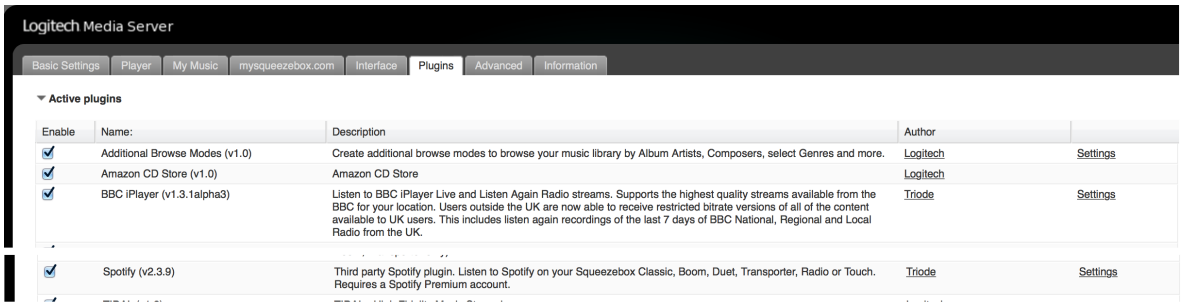


Figure 5: Non-Standard Active LMS Plug-ins Page (highlights)

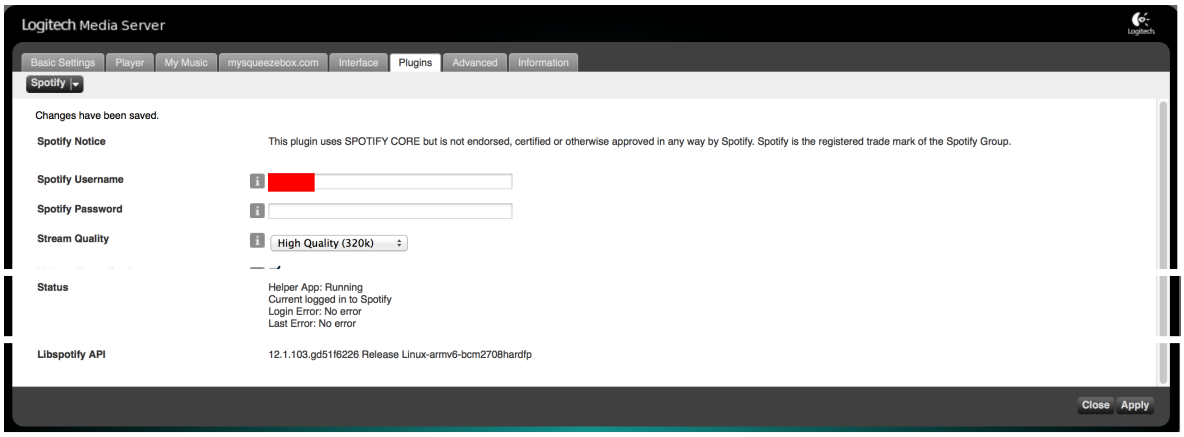


Figure 6: Spotify Plug-in Configuration Page (highlights)

It is necessary to configure LMS so that the 'official' Spotify plug-in (and all other Spotify plug-ins except the one mentioned next) are inactive. Then, Triode's 'third party' Spotify plug-in must be made active. This is all done in the **Plugins** tab of the LMS **Settings** page. See Figure 5 for the 'non-standard' plug-ins that should be active on the S-2020-S system.

Once 'installed', your Spotify Premium user name and password must be set up. To do that, press the `Settings` element on the right of the Spotify plug-in description in the LMS **Settings** page **Plugins** tab. This brings up the Spotify plug-in configuration page (see Figure 6).

Enter your Spotify Premium user name and password, then press the Apply button at the bottom right of the page. If successful, the Status section of the page should say:
Current logged in to Spotify

Controlling what Spotify streams is done with the LMS Spotify 'app'. This should be present in the LMS home page, as shown in Figure 7.

If **MyApps/Spotify** is not present, it will be necessary to restart LMS (using the piCorePlayer Main Page, LMS tab), the refresh the LMS home page. The 'app' should then be visible, as shown in Figure 7.

Once it is visible, simply click on it. The left hand half of the LMS home page will then allow you

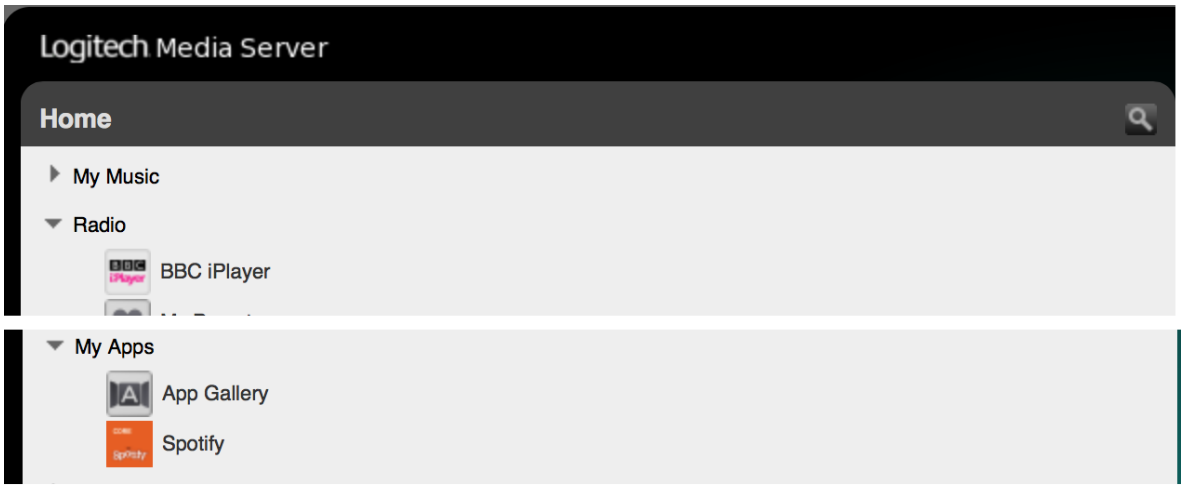


Figure 7: LMS Home Page (highlights)

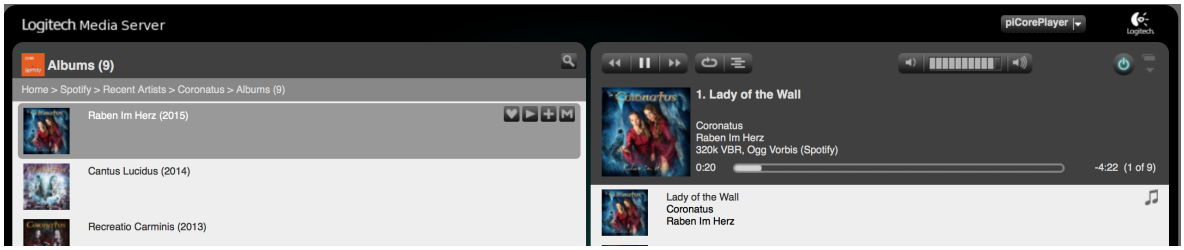


Figure 8: Playing an album in Spotify within LMS

to browse Spotify and select material to play. An example of this in action is shown in Figure 8.

In my experience, this service is very reliable indeed. I have yet to have it stop unexpectedly or otherwise misbehave.

2.5 Wi-Fi Diagnostics

It is sometimes useful to check the quality of the Wi-Fi signal being received by the S-2020-S. The easiest way to do this is to log in to the device via `ssh` — the command line Secure SHell program.

```
ssh tc@<IP-Address>
password: piCore
```

The following command gives useful information on the signal:

```
tc@piCorePlayer:~$ iwconfig wlan0
wlan0     IEEE 802.11bgn  ESSID:"HCCC_WLAN2"
          Mode:Managed  Frequency:2.442 GHz  Access Point: C4:3D:C7:C0:56:76
          Bit Rate=39 Mb/s   Tx-Power=1496 dBm
          Retry short limit:7   RTS thr:off   Fragment thr:off
          Power Management:off
          Link Quality=36/70  Signal level=-74 dBm
```

```
          Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
          Tx excessive retries:0  Invalid misc:0   Missed beacon:0

tc@piCorePlayer:~$
```

This signal level and link quality is perfectly adequate, by the way. The Raspberry Pi 3 built-in Wi-Fi gives no worse reception than other devices at the same location, either (in spite of some comments to that effect on the Internet).

However . . . it turns out that `ssh` access has a new wrinkle as of Ubuntu 16.04 and MacOS Sierra (10.12) . . . Just using `ssh` as shown gives:

```
Unable to negotiate with 192.168.1.141 port 22: no matching host key type found. Their offer: ssh-dss
```

The easiest long term way around this is to create (or edit) the file `~/.ssh/config` and add something along the following lines:

```
Host piCoreRed
     HostName 192.168.1.141
     HostKeyAlgorithms=+ssh-dss
```

This has started to happen because the DSA public key algorithm is now considered 'too weak'. See? Constant, unending, churn. For the best of reasons — but for devices with no public Internet access, this 'weakness' is not something to worry about. Obviously (I hope) devices like the S-2020-S should be insulated from the big, bad Internet by a router which is configured to prevent external access to them . . . until the router gets hacked, anyway . . .

Once `~/.ssh/config` has been set up along the lines shown, it will be possible to login with something along the lines of:

```
ssh tc@piCoreRed

Secure login powered by Dropbear SSH server on piCore.
tc@192.168.1.141's password:
```

The nameserver configuration can be checked with:

```
tc@piCorePlayer:~$ cat /etc/resolv.conf
nameserver 8.8.4.4
nameserver 8.8.8.8

tc@piCorePlayer:~$
```

Configuration files (e.g. to set the nameservers being used) can be edited directly using `vi`. For example:

```
tc@piCorePlayer:~$ sudo echo "nameserver 8.8.4.4" > /etc/resolv.conf
tc@piCorePlayer:~$ sudo echo "nameserver 8.8.8.8" >> /etc/resolv.conf
```

Whether such changes persist, I'm uncertain. The contents of `/etc/resolv.conf` seems to change unexpectedly, for instance — perhaps to values set in the router or other DHCP server.

2.6 Some Comments on the S-2020-S

These devices cost a little less than £150 each to put together. The most expensive item was the 256GB Flash memory device at a little over £50.

In theory, an S-2020-S could be used in conjunction with a smartphone and a rechargeable battery device as a portable music player! I haven't tried this yet, but it was a 'design goal' for the thing. Admittedly, it is a bit big for this — poacher's pockets would definitely be required to use it while walking around!

Whether it is a thing of beauty to look at or not is debatable. The plastic case is a practical necessity (for the internal Wi-Fi to work) but it might be looked down on. This particular case is very well made, though (for a plastic box). The lack of knobs and switches is a specific choice — physical access to the device is not required to use it — and it could be hidden away if the headphone socket is not being used. I think it has a stark attractiveness to it myself (but I would!).

Not much skill is required to put one of these things together. It is a matter of buying the various pre-assembled items it is comprised of, putting them in a box of some kind and installing the pre-built software. There is some software configuration, but it isn't hard (it is a little frustrating, potentially — it is essential to do things in a certain order, or things will not work, and you may have to start again from scratch).

The sound quality from the locally stored music library is, in my opinion, outstanding. Listening with Sennheiser HD600 headphones directly from the device[8] it is, I think, the best I have heard. In all cases, the volume control should be set at least 2 steps below maximum — some kind of internal clipping can occur if left at the maximum setting. I haven't investigated this, as sticking to the above rule avoids the problem entirely.

The non-BBC Internet radio services at 128 kb/s MP3 are very good (as good as an excellent FM tuner, I think, and vastly better than 128 kb/s DAB). They are also pretty reliable. Not perfectly so — but it is possible to listen to them for an entire day without interruptions.

The BBC iPlayer radio service is, at its best, superb. Live broadcasts on Radio 3 are terrific. It is quite fragile, though and you are lucky to get more than 2 hours[9] uninterrupted replay. There may be many reasons for this — not much about the Internet is guaranteed . . . but other services do much better. Without a very fast and robust DNS service, there are major quality issues. Having audible glitches every few tens of seconds is just unacceptable. Fortunately, using an appropriate DNS server really does seem to fix this — to my amazement.

Spotify Premium streaming at 320kb/s is very good indeed — it sounds as good as CDs to me, I think. It is also totally reliable.

The device should be maintainable fairly easily. The Raspberry Pi 3 and its power supply are very readily available and would be easy to replace today. Ten years from now, that may not be the

[8] Warning! The volume level for headphone listening must be adjusted to a much lower level than that for line output! This should be done before plugging the headphones in. The volume might be high enough to cause hearing damage if not reduced!

[9] With the Google DNS server and after moving DHCP duties off the router, I have had over 14 hours on occasion. It is hard to say what really makes a difference to this and what is just random.

case, but there almost certainly will be equivalents available. The IQaudIO piDAC+ may or may not be available in a few years time. I hope it will, but who knows? There is an alternative: the HifiBerry device, although that lacks the headphone amplifier. It seems likely that some equivalent would be available though. The USB memory stick will almost certainly remain available in to the fairly distant future — at ever higher capacities and/or lower costs. However, I expect the device to be very reliable — I hope it will still be working properly long after I am gone.

The S-2020-S device is an example of **the complete disconnect between sound quality and equipment cost that is possible with today's technologies**. It would be easy (if I had the money — which I freely admit I don't!) to spend several thousand pounds on each of the components the S-2020-S replaces and not get any better results as far as actual sound quality is concerned. In fact, it would be easy to spend vast sums and get worse results!

I personally can't see any reason to consider investing in any further source components in the future. Something close enough to perfection for me has been reached. Perhaps an attack of *audiophilia nervosa* will one day make me reconsider — but I doubt it!

3 The S-2020-P Pre-amplifier/Controller

The S-2020-S Universal Audio Source could be used with any competent amplifier and loudspeakers. It worked perfectly well with the Denon PMA-250UK and JPW Sonatas. But . . . I had grown used to being able to use a remote control to do just about everything with the Meridian G91. The Denon has no remote control capability. I found I really couldn't live without that remote control facility — I suppose I am just super lazy!

3.1 Background

I considered three possible ways to fix this problem:

- Replace the Denon with a classic ReVoX amplifier — the B251. I 'collect' ReVoX gear and have a matching B225 CD player and B795 turntable (almost never used). It was tempting, but, in the end, there seemed a few real drawbacks. Firstly, a good second hand B251 wasn't going to be cheap. Around £500 was a likely minimum and at that price, it would probably still need fettling: replacement of polarized capacitors, cleaning, etc. Fully refurbished ones are available for upwards of £1000, but I really wasn't prepared to spend that much. Secondly, as a result of the UK's decision to leave the EU (emphatically *not* my decision, I must say), B251s, typically to be found in Germany, were getting pricier by the day. Finally — and this was the killer — many B251s — even fully refurbished ones — do not come with a remote control! And the remote protocol is non-standard[10] . It might be that fancy Logitech Harmony universal remotes could deal with the protocol, but I wasn't certain. The 'real' remote is still available new — for over £130, though. A little reluctantly, I gave up on the B251 idea.

- Replace the Denon with a newer PMA model. These do have remote control and start at under £100 second hand. That would have been the easiest way to go, I think.

- Build a pre-amplifier which could be remote controlled. This wouldn't actually need to do any amplification, really. It would need to select input sources and control the volume with an attenuator. It would probably be better called a 'controller', in fact. I started looking at the hardware and software components that might be assembled to do this and concluded that it was definitely feasible. So . . . that was the way I chose to go.

The advantages of DIY audio have been described many times, I'm sure. For me, they are:

- You can get exactly the functionality you want, packaged the way you want it. No more. No less.

- You are in complete control of the components and assembly techniques used.

- You can spend as much time as you like on building it. Commercial considerations mean that there are always limits to how much time a company can spend building something — even if the price tag is allowed to be very high.

[10] There is also the issue that there are a lot more IR interference sources around now than there were in the early 1980s. In particular, the abominable compact fluorescent lamps — perhaps the worst technology ever forced on an unsuspecting public. Fortunately now giving way to LEDs. Modern IR receivers are designed to cope with this. My B225 CD player has had occasional troubles with CFLs, which is another reason to worry about the B251 route.

- You can get much better value for money with the DIY approach. Compared to some (insane) commercial offerings, nearly infinitely better, in fact!

There are some disadvantages, of course.

- Some construction techniques are infeasible to do at home or in very small volumes. Multi-layer PCBs and surface mount (at least with the smaller component sizes and more than two board layers) are virtually (or actually) impossible to use. Some things *need* these construction techniques to work — either at all, or with the desired level of performance. Most audio electronics doesn't, in my opinion — although some digital stuff does.

- Although it is possible to get very good value for money if aiming for a high (or very high) quality product, it isn't possible to build something as cheaply as 'commodity' products can be mass produced. Such 'commodity' products may be very good, too. I'm not sure a DIY equivalent of a 'low end' (but basically excellent) Denon PMA amplifier could be made for the price Denon charge, for example.

- Things that are theoretically feasible may simply require too much time and effort to do in practice — as a hobby, anyway. This may be the case for complex firmware, for example. Things have to be not just *possible* to do, but *easy enough* to do to make doing them a reasonable proposition. No doubt everyone has a different idea of what is 'easy enough'!

Many other people have designed there own pre-amplifiers and shared the details on the Internet. Two of these generous people influenced the S-2020-P design greatly. They are noted below.

3.2 Overview

The S-2020-P hardware has three main components:

- Passive input selector. By 'passive' I mean relays, of course. I have no doubt that semi-conductor switching can be made to work flawlessly, but relays are a viable option when only a small number will be required (for various reasons, the huge number that would be required for a large mixing console makes semiconductor switching virtually essential for them). Relays are easy to use and some of them are very small (by through hole component standards!).

- Passive volume control — again, this means one based on relays. A logarithmic ladder attenuator can be built where each step is a fixed number of decibels. A digital code word can set the attenuation in decibels — this is exactly what is needed for a volume control. Although related to R-2R ladder networks[11] used in many DACs, the logarithmic ladder attenuator wasn't invented until the 21st Century. See https://en.wikipedia.org/wiki/Logarithmic_resistor_ladder for the theory behind it. There is a website which has a calculator for designing these things: http://www.vaneijndhoven.net/jos/attenuator-calculator/index.html, which is amazingly helpful. A 6 bit attenuator with 1dB steps would cover the range 0 (maximum volume) to -63dB (minimum volume), which is a very satisfactory range. The default design example at Jos van Eijndhoven's site (above)

[11] First described in 1955.

does precisely this. No doubt semiconductor based designs (e.g. using programmable gain amplifiers) can also work flawlessly, but with 1 relay per bit, compact relays are viable for this application. There is no possibility of non-linear distortion with such a design — although there isn't really any possibility of significant non-linear distortion with properly designed semiconductor based solutions, either.

- Infra-red remote control and seven-segment LED status display based around an Arduino. Why an Arduino and not a naked micro-controller? There are several reasons:
  - There is a very capable multi-protocol IR control library available for the Arduino that was written by Ken Shirriff. This is described here: http://www.righto.com/2009/08/multi-protocol-infrared-remote-library.html and the code is available here: https://github.com/z3t0/Arduino-IRremote. This is a non-trivial piece of software and contains a great deal of difficult to acquire knowledge about the details of IR protocols. It is a major part of the software required for the S-2020-P.
  - There is a nice seven segment display library for the Arduino written by Dean Reading, and available from http://playground.arduino.cc/Main/SevenSegmentLibrary. Together with the IR protocol library, this is basically 'job done' for the software!
  - Between the relay control signals, the seven segment display control signals and the (single) IR receiver signal, quite a few pins are required on the micro-controller. The easiest approach is to choose a controller with enough (more than enough!) digital i/o pins — such as an Atmel Mega 2560. As far as I know, these only come in surface mount packages — which makes them problematic for use with a home etched PCB[12].

All in all, an Arduino Mega would seem the obvious way to go in the light of this. Add in very easy programming with no special tools required and, for me, it was a 'no brainer'. No doubt this approach will be looked down on in some circles, though. Although 'official' Arduino Mega devices are not cheap (about £45), 'clones' are available for much less — e.g. from Sainsmart[13] at £13.

3.3 Circuit Description

Figure 9 is the complete circuit diagram for the device. There are 5 inputs, any one of which may be switched to a 'monitor bus'. Any one of inputs 1 to 4 may also be switched independently to a second 'record bus', which is connected directly to a 'record' output socket pair. Input 5 is intended to be used for the 'reproduce' signal from a recorder/reproducer (i.e. what was traditionally a tape recorder, but which now may be any number of things).

The 'input bus' is connected to a unity gain buffer amplifier which serves two purposes:

- The input to the buffer has a low pass filter with -6dB at about 2MHz intended to reduce possible RF interference followed by a high pass filter which blocks DC and very low frequency signals. This is set too low really — the 47μF non-polarized capacitors just happened to be in my parts box!

[12] At least, with my skills, tools and experience.

[13] The Arduino is open hardware — it is neither illegal nor immoral to make a 'clone'. I do buy 'real' Arduinos too, but I have no qualms about using 'clones'.

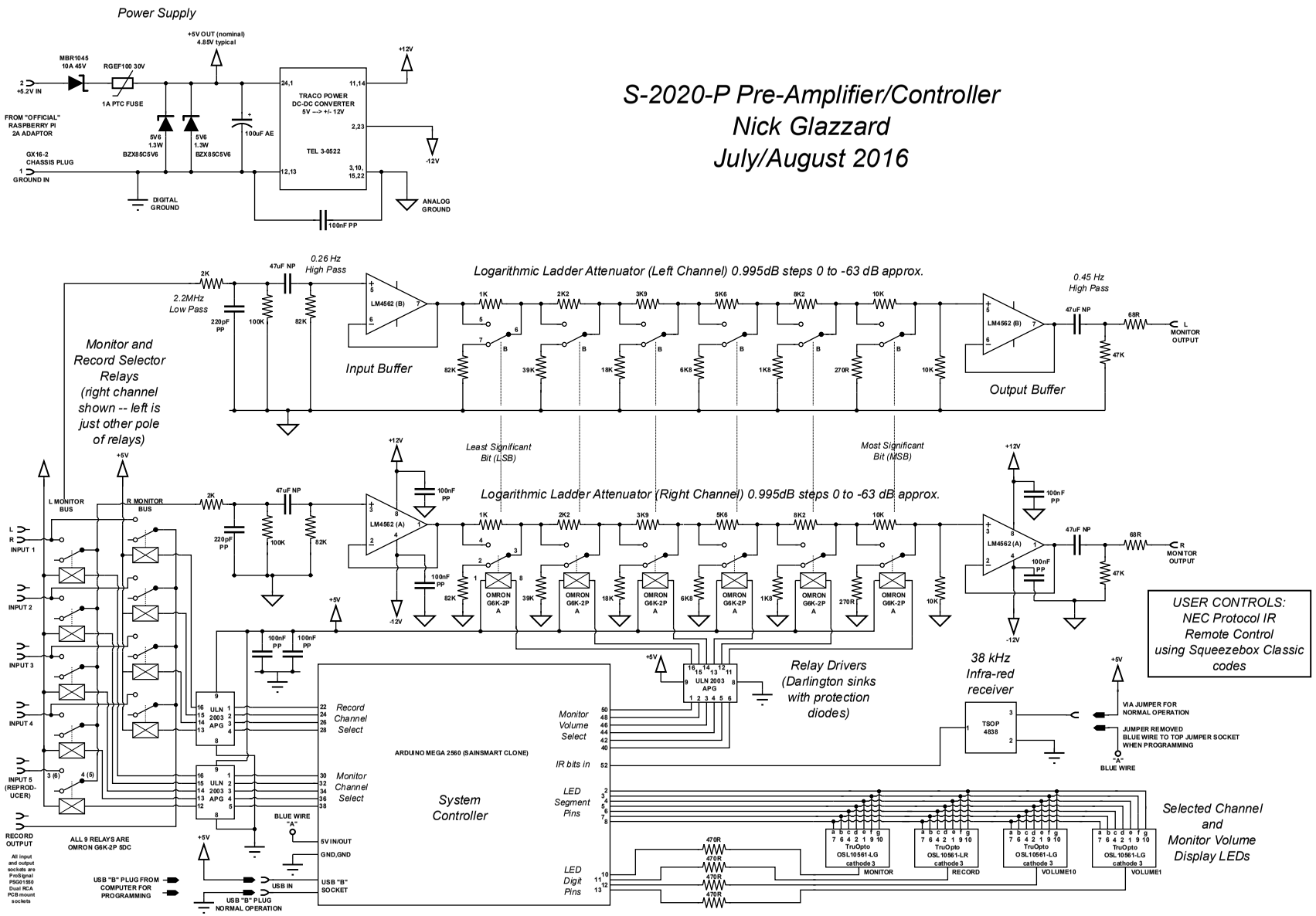


Figure 9: S-2020-P Circuit Diagram

- The output of the buffer amplifier is at a low impedance suitable for driving the passive logarithmic attenuator network.

The signal passes through the logarithmic attenuator network. This is a 'stereo' network — the same attenuation is applied to both channels. There is no provision for 'balance' adjustment. This is built with 1% tolerance resistors which will give much better tracking between channels than could be achieved with a traditional potentiometer — even very expensive ones.

The output of the attenuator (volume control) is buffered by another unity gain amplifier. This serves as the output line driver. The output has a DC blocking network and a series resistance which decouples the capacitance of any attached cables sufficiently to prevent ringing and possible oscillation issues.

This is all very straightforward, and was inspired partially by Mark Hennessy's design. See: http://www.markhennessy.co.uk/preamp/. I believe this is a well known project. It uses programmable gain amplifiers for volume control rather than a passive logarithmic attenuator network.

The control side of the device is carried out by an Arduino Mega 2560 card, mounted as a daughter board to the main PCB. This seems to work rather well mechanically. The relays are driven with ULN2003APG Darlington sinks. These contain internal protection diodes which appear across the relay coils to prevent damage to the driver transistors.

The 'user interface' to the device consists of an IR receiver and 4 seven segment displays — one of these displays the number of the input being monitored (listened to, range 1 to 5), one (a red one) displays the input being (potentially) recorded (range 1 to 4), and two display the volume. This is in the range 0 to 63 and is:

$$63 - attenuation$$

where $attenuation$ is the attenuation in decibels (pretty much exactly) set in the logarithmic attenuator network.

There are intentionally no traditional controls. Everything must be done with an infra-red remote control.

There are two PCBs in the device. Figures 10 and 11 show the copper and component sides of the 'main' board. This is a 220 mm by 100 mm board from Spiratronics which had a photo-resist coating applied — although there were problems that resulted in this coating not being used in the end. See Appendix D for more details.

The smaller 'front panel' PCB is 220 mm by 32 mm and was cut from a second Spiratronics board. The layout is shown in Figure 12. This carries the seven segment displays and the IR receiver. It is connected to the main board by a 14 way ribbon cable.

The seven segment displays are common cathode devices driven by multiplexed signals directly from the Arduino Mega. The TTL compatible output signal from the IR receiver goes directly to an Arduino input pin. Note that the IR receiver is a fairly sophisticated device — it contains filters

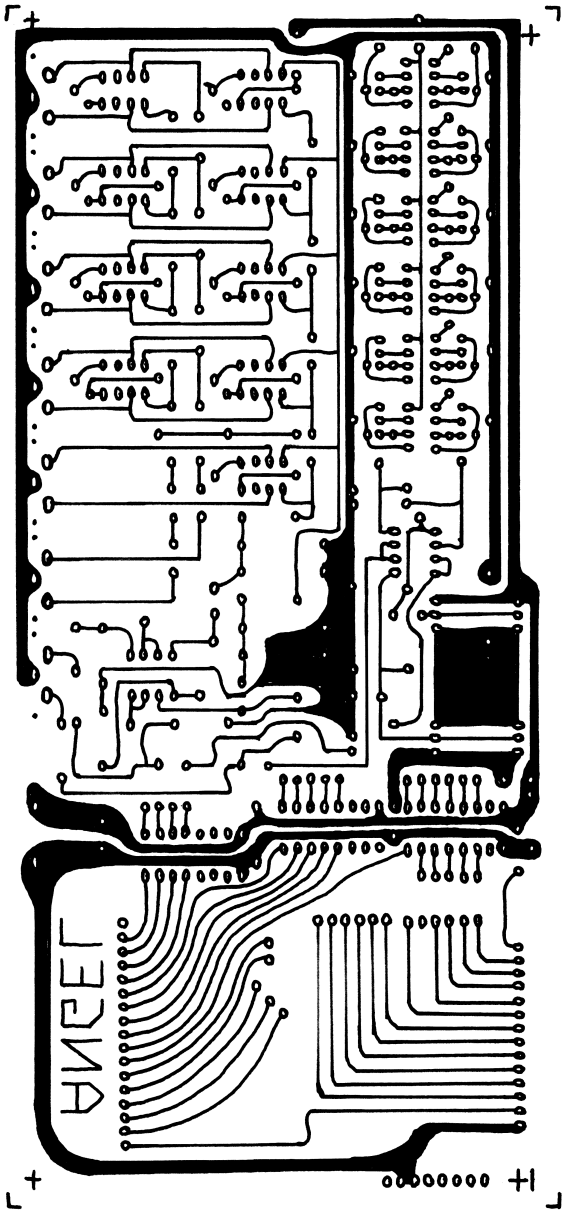


Figure 10: S-2020-P Main PCB – Copper Side

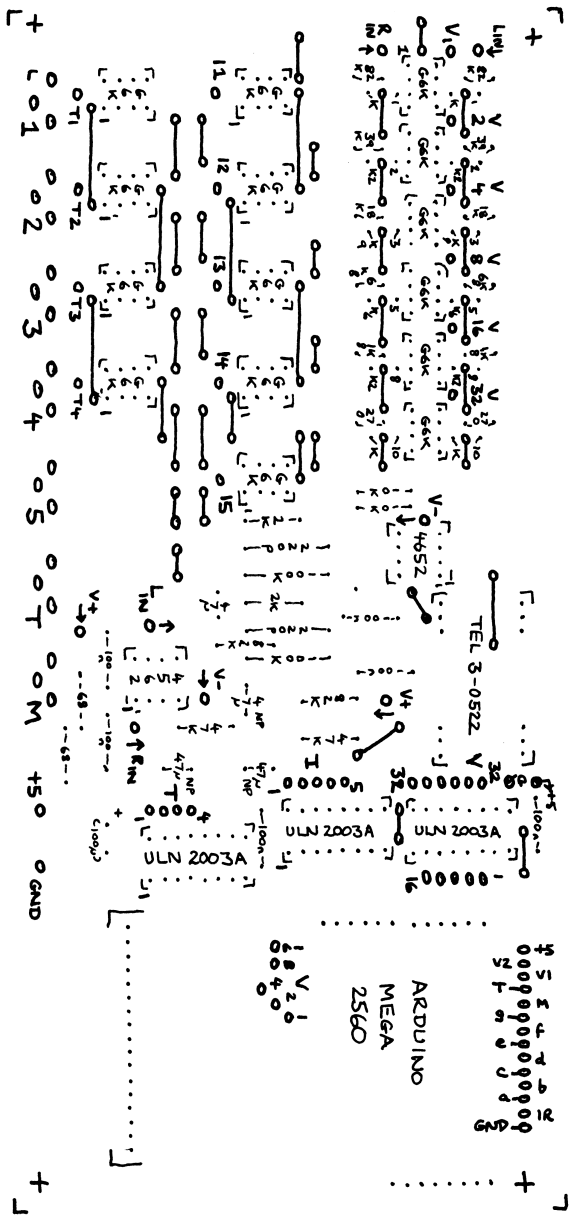


Figure 11: S-2020-P Main PCB – Component Side

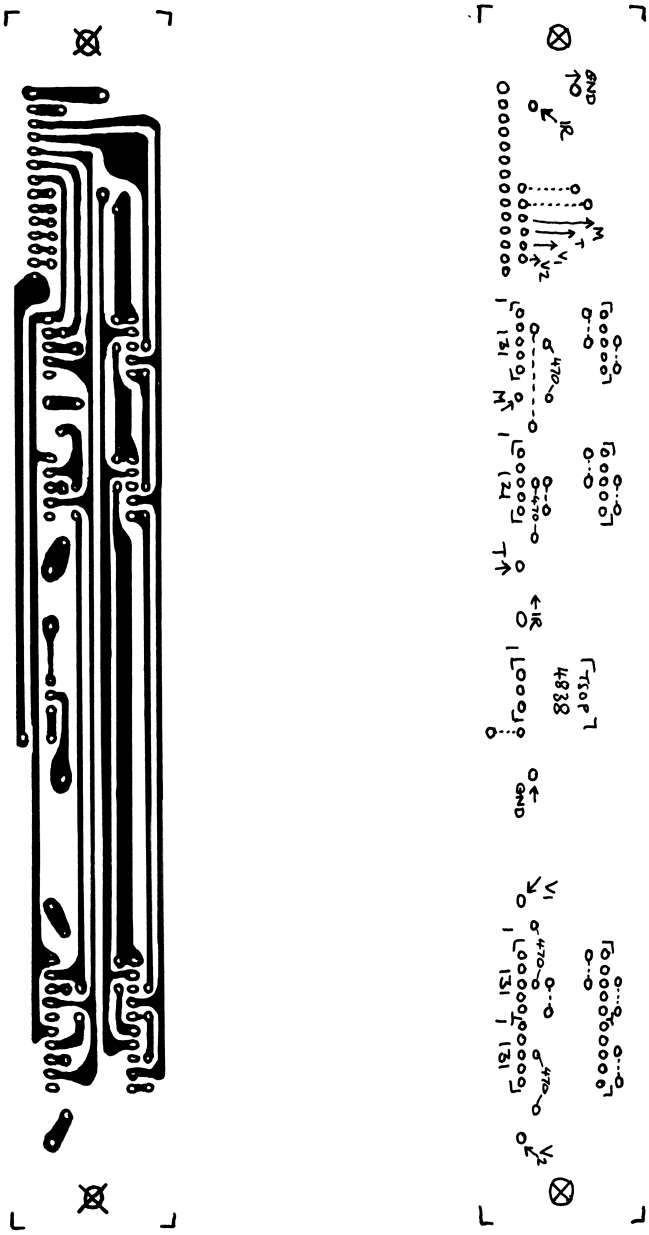


Figure 12: S-2020-P Front Panel PCB – Copper and Component Sides

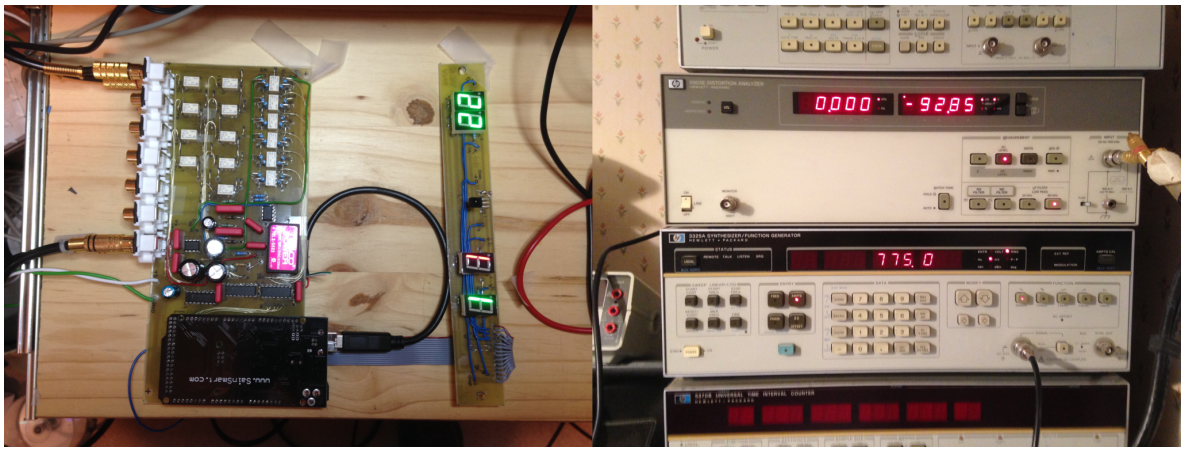


Figure 13: S-2020-P under test

intended to make reception robust even in the presence of rubbish thrown out by CFL lamps. Not bad for less than £0.90!

The power supply may be of interest. For compatibility with the S-2020-S device, an 'official' Raspberry Pi 2A power supply is used to supply 5.2V DC to the S-2020-P. This is fed through a fairly crude attempt to protect against reverse polarity, excessive current draw and over voltage which might or might not work in the unlikely event it was ever needed! The Arduino, relays, seven segment displays and IR receiver (i.e. all the 'digital' stuff) are fed directly from this power supply. The 5.2V (actually 4.85V after passage through the reverse polarity protection diode and PTC fuse) is converted to $\pm$12V for the buffer amplifiers by a Traco Power isolating DC-DC converter. There are completely separate analog and digital grounds, the only connection between which is a 100nF capacitor which I found reduced some indications of high frequency noise (far above the audio band) when measuring things with test equipment — but these indications may have been spurious, actually. This arrangement seems to work very well indeed.

Figure 13 shows the assembled PCBs of the S-2020-P under test. There was a period of panic when a 'wall wart' power supply was first used. Operation became unreliable — although it wasn't immediately obvious that the power supply was to blame. In fact, an Apple[14] 1A 5V power supply, when asked to deliver a mere 300mA, fell to 4.2V! Some bodges were made to the firmware to work around non-existent possible problems before I discovered the actual cause of the problem. This convinced me that the 'official' Raspberry Pi 2A power supply with its thick cables and 5.2V output was the only way to go. I haven't got around to removing the pointless firmware bodges yet. The firmware — bodges and all — is listed in Appendix C. It is very straightforward. Now the S-2020-P is in constant use and connected to everything else I can't work up the enthusiasm to get it out of the rack and re-program it! It works well enough as it is for my purposes — but it could be better as far as unwanted control repetitions due to holding down remote control buttons a little too long is concerned.

[14] I can hardly believe this is a genuine Apple device, actually. But, as far as I know, it is.

3.4 Firmware Development

Programming is very easy to do, actually. The software is developed using the standard Arduino IDE using what is apparently known as 'the Arduino language' — which is C++ with a few things predefined[15] . The Arduino IDE is started as follows on my Ubuntu development machine:

```
cd ~/arduino/arduino-1.0.5
./arduino
```

The IDE itself is written in Java, I believe. The projects are saved as 'sketches' in:

```
~/sketchbook/<PROJECT-FOLDER>/<PROJECT>.ino
```

Downloaded libraries (for the S-2020-P, `SevSeg` and `IRremote`) must be unzipped in to

```
~/arduino/arduino-1.0.5/libraries
```

and renamed to omit the `-master` postfix that comes from them being held in Github, after which they 'just work' in my experience.

The IDE must be informed that the target device is an Arduino Mega 2560 via the `Tools > Board` menu, otherwise uploads to the device will fail.

To upload firmware to the S-2020-P, it must be altered slightly from its 'normal operation' condition.

- The external power supply must be disconnected.
- The jumper on the front panel next to the TSOP-4838 must be removed.
- The 'blue wire' from the main board must be plugged in to the top pin of the socket from which the jumper was removed.
- The USB B plug which supplies external power to the Arduino daughter board must be unplugged.
- A USB cable (A plug to B plug) must be inserted to connect the Arduino daughter board to the development computer.

It should then be possible to upload firmware to the Arduino. This procedure guarantees there is no possible issue with power supplies and the Arduino. When programming is done, the normal operating condition must be restored by reversing the above steps.

3.5 Case Construction

Because the audio line inputs and outputs are on connectors attached directly to the main PCB — thereby fixing various dimensions — and also for aesthetic reasons, the S-2020-P was assembled in a custom built case. This was made from raw aluminium bar, aluminium plate and acrylic sheet. The result is shown (with the top removed) in Figure 14. The exterior aluminium surfaces were

[15] I've seen a few web sites that make a big deal of how confusing it is for the Arduino people to 'claim' they have their own language. I can't see this myself — it is pretty obvious that it is C++! I hadn't even realized anyone could think it wasn't until I chanced upon these sites.

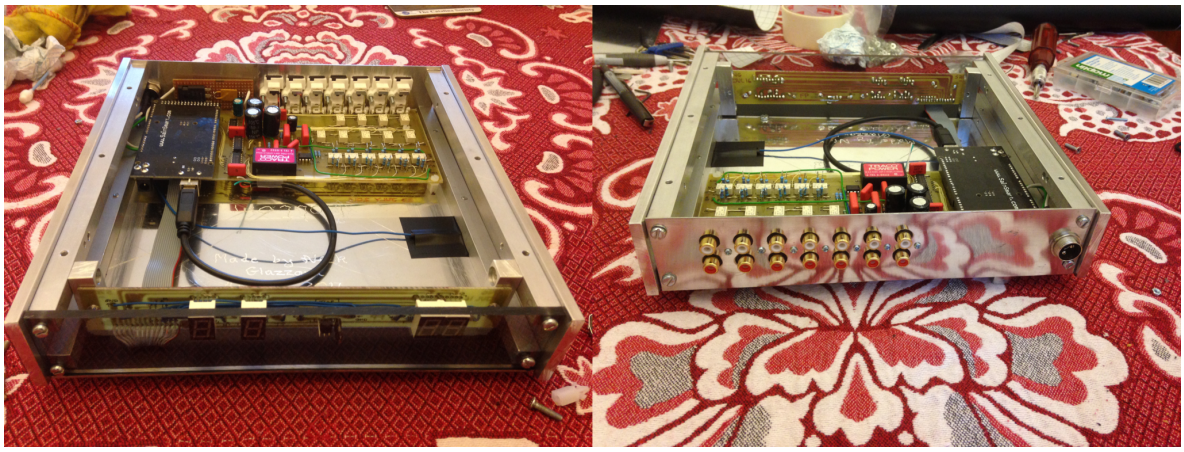


Figure 14: The S-2020-P custom case

polished with *Mothers Mag and Aluminium Polish*. The case was built with a hand hacksaw, a drill press, various drill bits, a centre punch, a set of needle files and a tap and die set. The result is quite satisfactory but it did take some time! And it isn't perfect. One thing I learned was that some 'play' must be built in to the exact front to back positioning of the main board — once the rear panel was drilled, the board had to be positioned so that the sockets were flush with it. With the tools and methods used, this required that the main board was mounted to the bottom panel with small slots in that panel rather than single holes — the accuracy required to avoid this is beyond me. All in all, though, the thing gives a suitable impression of quality and weight. Certainly not up to Dora Goodman's level of craftsmanship[16] but almost good enough. A bit more work to improve the fit and finish of the top and rear panels (the latter is too thin, actually) needs to be done — one day.

There is no attempt to disguise what the S-2020-P is — the opposite in fact. The materials and components from which it is made are clearly visible and unadorned. One of the joys of DIY is gaining an understanding of how things are built, what they are built from and how they work[17]. The 'aesthetics' of the S-2020-P case follow from that.

Some views of the completed S-2020-P device (moved out of the rack it normally lives in but still attached to all the other bits and pieces in the rack) are shown in Figure 15.

3.6 Remote Controls

Since the S-2020-P has no controls itself (not even a power switch), it is useless without a remote control. The remote chosen for this is the one that came with a Squeezebox Classic that (unfortunately) died — which sort of was the trigger to making major changes resulting in the S-2020 system being built. This is an NEC protocol remote which works well with the IRremote library for

[16] See https://www.instagram.com/doragoodman/ and despair — truly wonderful stuff.

[17] And what they cost to build — come to that. Now, the cost of the components for a DIY project is obviously not what people legitimately need to charge for a commercial product. With needing to pay people's wages, many and various 'overheads', marketing budgets and so on, a purchase price three times the component cost or a little more is perfectly reasonable. In some cases, commercial products are tremendous value for money. In other cases . . . well . . . draw your own conclusions.

Figure 15: The completed S-2020-P custom case

the Arduino.

Since one of the goals of S-2020 is maintainability, there is the issue of what would happen if this (quite elderly) remote broke. One possibility is to use the 'IR blaster' found in many tablets and smart phones with a remote control 'app'. For my Samsung Android tablet, *Smart IR Remote - AnyMote* is very effective. It lets you design your own remotes and, since it includes the Squeezebox Classic as one its predefined options, it is straightforward to start from this and set up just the buttons needed. The results of a 20 minute experiment in building a custom remote in this 'app' is shown with the original 'real' remote in Figure 16.

It would be tempting to build a new 'real' remote control, but this would require a low power design. Standard Arduinos are not suitable for this, and more work is needed to identify what might be. I doubt I will get around to building this. *But*, being slightly nuts, I did . . . . See Section F.

Keys on the current 'real' remote are used as follows:

- [1] to [5] select the input to be monitored (listened to).
- [6] to [9] select the input to record — 6 for input 1 to 9 for input 4.
- [0] mutes and un-mutes.

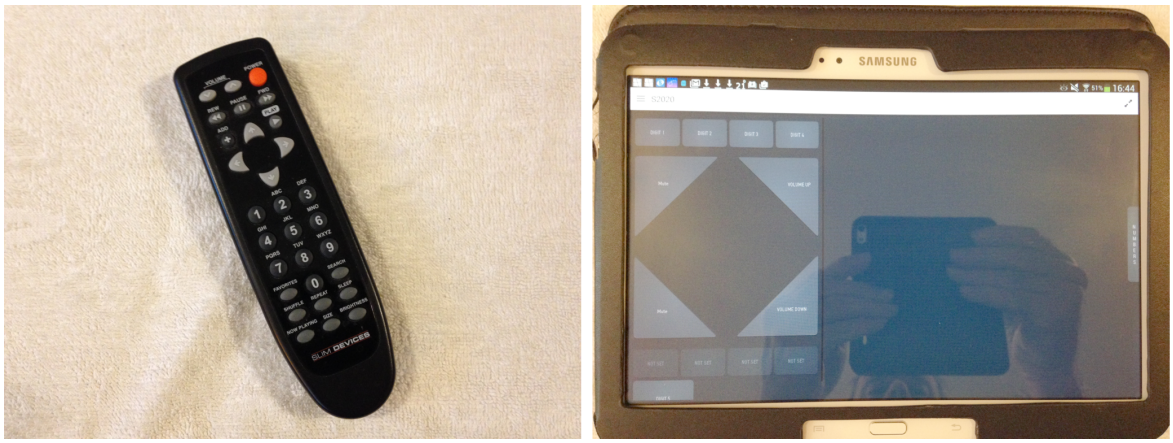


Figure 16: Real and virtual S-2020-P remote controls

- The right and left arrow keys cycle through the inputs for monitoring.
- The up and down arrow keys change the display brightness — although only the step between the lowest brightness and the next makes much difference!
- The power key mutes and sets the volume to 0. It is pretty useless, actually.

The exact codes transmitted for each key, and all other details, can be found from the firmware listing in Appendix C.

Note that the S-2020-P is not intended to be powered off in normal use. It uses less than 300mA or 1.5W, contains nothing that generates significant heat and I would expect it to be highly reliable[18].

Figure 17 shows both the S-2020-P and S-2020-S in the equipment rack where they belong.

[18]Of course, only time will tell.

Figure 17: The S-2020-P and S-2020-S in their proper place

A Setting up a DHCP server on Ubuntu Server

My router's DHCP server's inability to actually map 'reserved' MAC addresses to constant IP addresses — in spite of it promising to do so — rapidly became too annoying when using the S-2020-S devices. So I installed a DHCP server on my Ubuntu Server machine. It may be that a better solution would be to use the Dnsmasq program, as that would handle name to IP address mapping too. But the approach I took seems to work well for pure DHCP. One reason not to use Dnsmasq was that I didn't want to potentially mess up fast DNS resolution that finally seemed to be working. All would probably be OK, but it is another component in the DNS resolution chain (perhaps). Fast DNS resolution is so critical to proper operation of the BBC iPlayer plug-in that I didn't want to risk doing anything that might upset it!

To install the DHCP server:

```
sudo apt-get install isc-dhcp-server
```

There are then two configuration files to edit. The network interface `eth0` must be set in `/etc/default/isc-dhcp-server`. It is genuinely obvious how to do this! Then the main configuration file: `/etc/dhcp/dhcpd.conf` must edited. This is the contents of mine (which seems to be working):

```
#
# Sample configuration file for ISC dhcpd for Debian
#
# Attention: If /etc/ltsp/dhcpd.conf exists, that will be used as
# configuration file instead of this file.
#
#

# The ddns-updates-style parameter controls whether or not the server will
# attempt to do a DNS update when a lease is confirmed. We default to the
# behavior of the version 2 packages ('none', since DHCP v2 didn't
# have support for DDNS.)
ddns-update-style none;

default-lease-time 600;
max-lease-time 7200;

# If this DHCP server is the official DHCP server for the local
# network, the authoritative directive should be uncommented.
authoritative;

# Use this to send dhcp log messages to a different log file (you also
# have to hack syslog.conf to complete the redirection).
log-facility local7;

subnet 192.168.1.0 netmask 255.255.255.0 {
  range 192.168.1.100 192.168.1.199;
  #option domain-name-servers ns1.internal.example.org;
  #option domain-name "internal.example.org";
  option routers 192.168.1.1;
  option broadcast-address 192.168.1.255;
  default-lease-time 86400;
  max-lease-time 259200;
}

host piCorePlayer {
  hardware ethernet b8:27:eb:26:57:76;
  fixed-address 192.168.1.140;
}

host piCoreRed {
  hardware ethernet b8:27:eb:43:fe:e9;
  fixed-address 192.168.1.141;
}
```

The router's DHCP server was then turned off and the new DHCP server started.

```
sudo service isc-dhcp-server restart
```

After rebooting the S-2020-S devices, they come up on the expected IP addresses: `.140` for piCorePlayer and `.141` for piCoreRed — thankfully! If nothing else, these can be bookmarked in browsers to allow easy reliable access to the devices.

B A script to make S-2020-S library updates more convenient

Assuming that CDs are ripped on an Ubuntu Linux machine using the tools and methods outlined in Section 2.1, there is a Bash script that automates the transfer of audio files to the S-2020-S devices and also adds scanned cover artwork to the FLAC files. The full script is shown below.

```
#!/bin/bash
echo "upalbum: use in Music/artist directory."
if [ "$#" -ne 2 ]; then
    echo "Usage: upalbum albumdir coverscan.jpg"
    exit 1
fi
albumdir=$1
scanjpg=$2
echo $albumdir $scanjpg
if [ ! -f $scanjpg ]; then
    echo "$scanjpg does not exists as a regular file."
    exit 2
fi
if [ ! -d "$albumdir" ]; then
    echo "$albumdir does not exists as a directory."
    exit 3
fi
# Insert cover scan
cd "$albumdir"
metaflac --import-picture-from=../${scanjpg} *.flac
cd ..
# Copy the result
scp -r "$albumdir" tc@192.168.1.140:/mnt/audiolib
scp -r "$albumdir" tc@192.168.1.141:/mnt/sda1/audiolib
echo "Done."
exit 0
```

This script should be placed somewhere on the `PATH` — e.g. `/usr/local/bin`

As an example, given that a new CD had been ripped (with Sound Juicer) into `~/Music/Anartist/Somealbum` and the the CD cover has been scanned (with SimpleScan) into `~/Music/Anartist/somealbum.jpg`, then the commands:

```
cd ~/Music/Anartist
upalbum.sh Somealbum somealbum.jpg
```

will first insert the cover art in to each FLAC file in `Somealbum`, then copy the whole of `Somealbum` to both S-2020-S devices.

A very similar script for use with single FLAC files originating from home made recordings and using a single generic image for all files is `uprecording.sh`:

```
cd ~/flacs
uprecording.sh playingGuitar.flac
```

The copy processes will require the entry of the password for user `tc` on the S-2020-S devices (`piCore`). This nuisance can be avoided by setting up authentication for `ssh` using public keys. Assuming you already have generated a public/private key pair on the machine used for ripping, the S-2020-S devices can be configured once to accept public key authentication for your account on the ripping machine. To do this, first ensure there is a `.ssh` directory in the home folder for `tc` on the S-2020-S device.

```
$ubuntu: ssh tc@192.168.1.141
password: piCore
$picore: mkdir -p .ssh
$picore: exit
$ubuntu:
```

Then transfer the public key for your account (on the Ubuntu machine) to the authorized key list (for user `tc`) on the S-2020-S device:

```
$ubuntu: cd ~
$ubuntu: cat .ssh/id_rsa.pub | ssh tc@192.168.1.141 'cat >> .ssh/authorized_keys'
$ubuntu:
```

No password should then be required when logging in to the S-2020-S as `tc` from the Ubuntu machine (when you are using your account on that Ubuntu machine):

```
$ubuntu: ssh tc@192.168.1.141

Secure login powered by Dropbear SSH server on piCore.
        _ -----             ___  __
   ___  (_) ___/__  _______ / _ \/ /__ ___ _____ ____
  / _ \/ / /__/ _ \/ __/ -_) ___/ / _ `/ // / -_) __/
 / .__/_/\___/\___/_/  \__/_/  /_/\_,_/\_, /\__/_/
/_/                                    /___/

piCorePlayer = piCore + Squeezelite + Raspberry Pi

The software is provided "as is" without warranty of any kind,
either express or implied, including without limitation any implied
warranties of condition, uninterrupted use, merchantability,
fitness for a particular purpose, or non-infringement.
tc@piCoreRed:~$
```

This will allow the copy processes in `upalbum.sh` to work without needing your intervention (typing in the password for the S-2020-S `tc` account).

B.1 Making a public/private key pair

If you do not yet have a public/private key pair for your account on the Ubuntu machine used for ripping, the pair can be created using:

```
cd ~
ssh-keygen -t rsa
(no passphrase)
```

This makes the file `.ssh/id_rsa.pub`, which is the public part of the public/private key pair. *You should not do this if you already have a key pair!*

B.2 One way of adding S-2020-S devices to hosts files

One way of adding a record for an S-2020-S device (or anything, actually) to `/etc/hosts` without needing to open an editor with the right permissions is this:

```
sudo cat "192.168.1.140<CTRL-V><CTRL-I>piCorePlayer" >> /etc/hosts
```

where `<CTRL-V>` means hold down the control key and press the `V` key, etc. This enters a literal tab which goes in to the `/etc/hosts` file. *BEWARE:* using > instead of » will obliterate `/etc/hosts`!

C S-2020-P Firmware Listing

```
//
// Arduino Mega 2560 firmware for S-2020-P pre-amplifier/controller
// Nick Glazzard, July and August 2016.
//

#include <IRremote.h>
#include <SevSeg.h>

// Infra-red receiver input pin.
const int IR_PIN = 52;

// 7 segment display segment pins.
const int SEG_A_PIN = 2;
const int SEG_B_PIN = 3;
const int SEG_C_PIN = 4;
const int SEG_D_PIN = 5;
const int SEG_E_PIN = 6;
const int SEG_F_PIN = 7;
const int SEG_G_PIN = 8;
const int SEG_P_PIN = 23; // Decimal point. Not used.

// 7 segment display digit pins.
const int DIGIT_M_PIN = 10;
const int DIGIT_T_PIN = 11;
const int DIGIT_V1_PIN = 12;
const int DIGIT_V2_PIN = 13;

// Tape source select pins.
const int TAPE_1_PIN = 22;
const int TAPE_2_PIN = 24;
const int TAPE_3_PIN = 26;
const int TAPE_4_PIN = 28;

// Monitor source select pins.
const int MONITOR_1_PIN = 30;
const int MONITOR_2_PIN = 32;
const int MONITOR_3_PIN = 34;
const int MONITOR_4_PIN = 36;
const int MONITOR_5_PIN = 38;

// Volume control pins.
const int VOLUME_32_PIN = 40;
const int VOLUME_16_PIN = 42;
const int VOLUME_8_PIN = 44;
const int VOLUME_4_PIN = 46;
const int VOLUME_2_PIN = 48;
const int VOLUME_1_PIN = 50;

// IR remote command values.
// These are values produced by the NEC protocol Slim Device remote control.
const unsigned long MONITOR_CHANNEL_0 = 0x76899867; // 0 key. Off / mute.
const unsigned long MONITOR_CHANNEL_1 = 0x7689f00f; // 1 key
const unsigned long MONITOR_CHANNEL_2 = 0x768908f7; // 2 key.
const unsigned long MONITOR_CHANNEL_3 = 0x76898877; // 3 key.
const unsigned long MONITOR_CHANNEL_4 = 0x768948b7; // 4 key.
const unsigned long MONITOR_CHANNEL_5 = 0x7689c837; // 5 key

const unsigned long TAPE_CHANNEL_1 = 0x768928d7; // 6 key
const unsigned long TAPE_CHANNEL_2 = 0x7689a857; // 7 key.
const unsigned long TAPE_CHANNEL_3 = 0x76896897; // 8 key.
const unsigned long TAPE_CHANNEL_4 = 0x7689e817; // 9 key.

const unsigned long VOLUME_UP = 0x7689807f; // Volume up key.
const unsigned long VOLUME_DOWN = 0x768900ff; // Volume down key.

const unsigned long RESET_ZERO = 0x768940bf; // Power key.

const unsigned long ARROW_UP = 0x7689e01f; // Brightness up.
const unsigned long ARROW_DOWN = 0x7689b04f; // Brightness down.

const unsigned long ARROW_LEFT = 0x7689906f; // Monitor channel down.
const unsigned long ARROW_RIGHT = 0x7689d02f; // Monitor channel up.


// Create Infra-red receiver and 7 segment display objects.
IRrecv irrecv(IR_PIN);
SevSeg sevseg;

// Globals for the volume, tape and monitor control words.
// These are bit fields.
byte volume_word = 0;
byte tape_word = 1;
byte monitor_word = 1;

// Selected channel values.
byte tape_channel = 1;
```

```
byte monitor_channel = 1;

// Global for the display word. This is a single integer.
int display_word = 0;

// Keep track of being muted separately.
bool muted = true;
int pre_mute_channel = 1;

// Display brightness
int brightness = 90;

unsigned long last_obeyed_time;

void update_display_word( void )
//------------------------------
// Form a display word from the volume_word, tape and monitor channels values.
{
  if( muted )
    display_word = 10000;
  else
    display_word = 1000 * monitor_channel + 100 * tape_channel + volume_word;
}

void set_monitor_word( byte channel )
//-----------------------------------
// Set one bit in the monitor word, and set the relay states.
{
  // Convert channel number to bit field.
  if( channel == 0 )
    monitor_word = 0;
  else
    monitor_word = ( 1 << (channel-1) );

  // Turn off all channels.
  digitalWrite(MONITOR_1_PIN,LOW);
  digitalWrite(MONITOR_2_PIN,LOW);
  digitalWrite(MONITOR_3_PIN,LOW);
  digitalWrite(MONITOR_4_PIN,LOW);
  digitalWrite(MONITOR_5_PIN,LOW);

  // Turn on the first (and only) set channel.
  if( (monitor_word & 1)  != 0 )digitalWrite(MONITOR_1_PIN,HIGH);
  if( (monitor_word & 2)  != 0 )digitalWrite(MONITOR_2_PIN,HIGH);
  if( (monitor_word & 4)  != 0 )digitalWrite(MONITOR_3_PIN,HIGH);
  if( (monitor_word & 8)  != 0 )digitalWrite(MONITOR_4_PIN,HIGH);
  if( (monitor_word & 16) != 0 )digitalWrite(MONITOR_5_PIN,HIGH);
}

void set_tape_word( byte channel )
//--------------------------------
// Set one bit in the tape word, and set the relay states.
{
  // Convert channel number to bit field.
  if( channel == 0 )
    tape_word = 0;
  else
    tape_word = ( 1 << (channel-1) );

  // Turn off all channels.
  digitalWrite(TAPE_1_PIN,LOW);
  digitalWrite(TAPE_2_PIN,LOW);
  digitalWrite(TAPE_3_PIN,LOW);
  digitalWrite(TAPE_4_PIN,LOW);

  // Turn on the first (and only) set channel.
  if( (tape_word & 1)  != 0 )digitalWrite(TAPE_1_PIN,HIGH);
  if( (tape_word & 2)  != 0 )digitalWrite(TAPE_2_PIN,HIGH);
  if( (tape_word & 4)  != 0 )digitalWrite(TAPE_3_PIN,HIGH);
  if( (tape_word & 8)  != 0 )digitalWrite(TAPE_4_PIN,HIGH);
}

void set_volume( void )
//---------------------
// Change the volume to the value now in the volume_word.
// This simple approach works better than alternatives which reset to the minimum of the current and new,
// use delays and so on.
{
  // Turn all off from high to low.
  digitalWrite(VOLUME_32_PIN,LOW);
  digitalWrite(VOLUME_16_PIN,LOW);
  digitalWrite(VOLUME_8_PIN, LOW);
  digitalWrite(VOLUME_4_PIN, LOW);
  digitalWrite(VOLUME_2_PIN, LOW);
  digitalWrite(VOLUME_1_PIN, LOW);

  // Turn on from low to high.
  digitalWrite(VOLUME_1_PIN,(((volume_word & 1)!=0)?HIGH:LOW));
  digitalWrite(VOLUME_2_PIN,(((volume_word & 2)!=0)?HIGH:LOW));
  digitalWrite(VOLUME_4_PIN,(((volume_word & 4)!=0)?HIGH:LOW));
```

```
  digitalWrite(VOLUME_8_PIN,(((volume_word & 8)!=0)?HIGH:LOW));
  digitalWrite(VOLUME_16_PIN,(((volume_word & 16)!=0)?HIGH:LOW));
  digitalWrite(VOLUME_32_PIN,(((volume_word & 32)!=0)?HIGH:LOW));
}

void setup()
{
  //Serial.begin(9600);

  // Set the initial state of the three control words again, just in case!
  volume_word = 0;
  tape_word = 1;
  monitor_word = 1;

  // And the display word. And muted state.
  display_word = 0;
  muted = true;

  // And the currently selected channels.
  tape_channel = 1;
  monitor_channel = 1;

  // And the display brightness.
  brightness = 90;

  // Initialize last obeyed command time.
  last_obeyed_time = millis();

  // Set up the tape relay control pins.
  pinMode(TAPE_1_PIN,OUTPUT);
  pinMode(TAPE_2_PIN,OUTPUT);
  pinMode(TAPE_3_PIN,OUTPUT);
  pinMode(TAPE_4_PIN,OUTPUT);

  // Set up the monitor relay control pins.
  pinMode(MONITOR_1_PIN,OUTPUT);
  pinMode(MONITOR_2_PIN,OUTPUT);
  pinMode(MONITOR_3_PIN,OUTPUT);
  pinMode(MONITOR_4_PIN,OUTPUT);
  pinMode(MONITOR_5_PIN,OUTPUT);

  // Set up the volume controller relay control pins.
  pinMode(VOLUME_32_PIN,OUTPUT);
  pinMode(VOLUME_16_PIN,OUTPUT);
  pinMode(VOLUME_8_PIN,OUTPUT);
  pinMode(VOLUME_4_PIN,OUTPUT);
  pinMode(VOLUME_2_PIN,OUTPUT);
  pinMode(VOLUME_1_PIN,OUTPUT);

  // Set the initial value of the display word.
  update_display_word();

  // Set up the 7 segment display object.
  byte num_digits = 4;
  byte digit_pins[] = {DIGIT_M_PIN,DIGIT_T_PIN,DIGIT_V1_PIN,DIGIT_V2_PIN};
  byte segment_pins[] = {SEG_A_PIN,SEG_B_PIN,SEG_C_PIN,SEG_D_PIN,SEG_E_PIN,SEG_F_PIN,SEG_G_PIN,SEG_P_PIN};
  sevseg.begin(COMMON_CATHODE,num_digits,digit_pins,segment_pins);
  sevseg.setBrightness( brightness );

  // Start the IR receiver.
  irrecv.enableIRIn();
}

void loop()
{
  // Record the currently selected channel values and the volume on entry.
  byte entry_volume_word = volume_word;
  byte entry_tape_channel = tape_channel;
  byte entry_monitor_channel = monitor_channel;
  bool entry_muted = muted;

  // Check for an IR remote command being received.
  bool got_value = false;
  decode_results ir_data;
  if( irrecv.decode( &ir_data ) ){

    // See how long since we last obeyed a command.
    // Only execute a command if some small time has passed since the last obeyed command.
    // This is a balance between responsiveness and "key bounce" ... it depends on the remote
    // key characteristics and the user.
    unsigned long time_passed = millis() - last_obeyed_time;
    if( time_passed > 150 ){

      // Respond to the command.
      switch( ir_data.value ){
        case MONITOR_CHANNEL_0:// 0 key. Off / mute.
          muted = ! muted;
          if( muted )
            monitor_channel = 0;
          else
```

```
      monitor_channel = pre_mute_channel;
    set_monitor_word( monitor_channel );
    break;
  case MONITOR_CHANNEL_1: // 1 key
    monitor_channel = 1;
    pre_mute_channel = monitor_channel;
    set_monitor_word( monitor_channel );
    break;
  case MONITOR_CHANNEL_2: // 2 key.
    monitor_channel = 2;
    pre_mute_channel = monitor_channel;
    set_monitor_word( monitor_channel );
    break;
  case MONITOR_CHANNEL_3: // 3 key.
    monitor_channel = 3;
    pre_mute_channel = monitor_channel;
    set_monitor_word( monitor_channel );
    break;
  case MONITOR_CHANNEL_4: // 4 key.
    monitor_channel = 4;
    pre_mute_channel = monitor_channel;
    set_monitor_word( monitor_channel );
    break;
  case MONITOR_CHANNEL_5: // 5 key
    monitor_channel = 5;
    pre_mute_channel = monitor_channel;
    set_monitor_word( monitor_channel );
    break;
  case TAPE_CHANNEL_1: // 6 key
    tape_channel = 1;
    set_tape_word( tape_channel );
    break;
  case TAPE_CHANNEL_2: // 7 key.
    tape_channel = 2;
    set_tape_word( tape_channel );
    break;
  case TAPE_CHANNEL_3: // 8 key.
    tape_channel = 3;
    set_tape_word( tape_channel );
    break;
  case TAPE_CHANNEL_4: // 9 key.
    tape_channel = 4;
    set_tape_word( tape_channel );
    break;
  case VOLUME_UP: // Volume up key.
    if( volume_word < 63 ){
      ++volume_word;
      set_volume();
    }
    break;
  case VOLUME_DOWN: // Volume down key.
    if( volume_word > 0 ){
      --volume_word;
      set_volume();
    }
    break;
  case RESET_ZERO: // Power key.
    tape_channel = 1;
    monitor_channel = 1;
    pre_mute_channel = 1;
    volume_word = 0;
    set_tape_word( tape_channel );
    set_monitor_word( monitor_channel );
    set_volume();
    break;
  case ARROW_UP: // Increase brightness.
    if( brightness < 99 ){
      brightness += 10;
      if( brightness > 99 )
        brightness = 99;
      sevseg.setBrightness(brightness);
    }
    break;
  case ARROW_DOWN: // Decrease brightness.
    if( brightness > 0 ){
      brightness -= 10;
      if( brightness < 0 )
        brightness = 0;
      sevseg.setBrightness(brightness);
    }
    break;
  case ARROW_LEFT: // Decrement monitor channel.
    if( monitor_channel > 1 ){
      --monitor_channel;
      set_monitor_word( monitor_channel );
    }
    break;
  case ARROW_RIGHT:
    if( monitor_channel < 5 ){
      ++monitor_channel;
```

```
            set_monitor_word( monitor_channel );
          }
          break;
      }

      last_obeyed_time = millis();
    } // time passed.

    got_value = true;
  }

  // If the entry values have been changed, update the display word.
  if( (entry_volume_word != volume_word) ||
      (entry_tape_channel != tape_channel) ||
      (entry_monitor_channel != monitor_channel) ||
      (entry_muted != muted) )
      update_display_word();

  // Display the display word.
  //Serial.print("Display word: ");Serial.print(display_word,DEC);Serial.println("");
  sevseg.setNumber( display_word, 0 );
  sevseg.refreshDisplay();

  if( got_value )
    irrecv.resume();
}
```

Figure 18: UV exposure unit, master and test strip

D S-2020-P Printed Circuit Board and other notes

Although I have made many printed circuits boards at home I couldn't say that I have developed *an absolutely reliable* means of doing it. I started with toner transfer — laser printing on to shiny paper and then ironing the result on to the copper side of the board. I reckon I got about a 30% success rate with this.

This was so frustrating that I invested in a cheap UV exposure unit and went down the photo-resist route. Using pre-coated boards from Spiratronics I got a 100% success rate throughout the RRI project (which had many complex boards). I printed the designs on 0.1 mm thick laser printable overhead transparency sheets using an OKI C5650 printer. Stacking three such sheets on top of one another — perfectly aligned — gave dense enough blacks to block UV. Figure 18 shows the Mega Electronics LV202E UV exposure unit, the triple stacked laser printed pattern to be transferred to the photo-resist and an etched 'exposure test strip' that was done to determine the optimum exposure time for the board and developer being used. It also shows the exposure unit closed with labels giving instructions on how to use it.

But — it turned out that this 'exposure strip' was not appropriate. Unfortunately, the boards I ordered this time from Spiratronics were not the same as those used in the RRI project. Using the exposure parameters used before, the pattern in the resist on the new board was very thin — useless, in fact. Note that this is not a fault with the board — the exposure time simply needed to be 'calibrated' for that particular board type. Actually, changing the board type (and hence the resist it is coated with), the developer or the etching process will have an effect on the success (or otherwise) of the final PCB. For the UV process, I use the Seno range of applicators (110 developer and 120 resist stripper) which I have found very effective and convenient. Unfortunately, I believe they have been discontinued.

There are definite limits to what I have tried with home made PCBs. I have never attempted to tin the etched boards. Tinning is pretty, but I don't regard it as necessary if the boards are stuffed and soldered within a day or two of being etched. I have never attempted to make double sided boards or boards with tracks less than 0.3 mm wide, or bigger than the 220 mm by 100 mm used in the S-2020-P. I regard these as design limits since I don't think I would have a high success

Figure 19: Press-N-Peel paper, before and after use, and laser printable adhesive film

rate trying to make boards that exceed them.

All of the boards in the RRI project (well, almost all) were designed with Kicad. This involves going from schematic capture, through adding new components to component libraries to routing the boards. Kicad is an excellent piece of open source software, but using it is a fairly 'heavy weight' process.

Other small projects I have done by hand, just drawing out the traces on $\frac{1}{10}th$ inch graph paper, then tracing over that with ink on to tracing paper, then scanning this in and cleaning it up (where needed) with image processing software. This is the way I went for the S-2020-P — although it is probably at the limits in size for this method. The main reason was that I didn't want to try to define the Arduino Mega as a component! Nor the G6K relays, to be honest. So — if you thought the PCBs looked hand drawn, they were!

Since the UV process didn't work out, I stripped the resist off the boards completely and used *Press–N–Peel* to transfer the design to the copper. This costs about £20 for 5 sheets, so it is fairly expensive. Basically, it is a properly engineered version of toner transfer. The design is printed with a laser printer on to a *Press–N–Peel* sheet and then the result is ironed on to the board to create an etch resistant barrier where tracks are present. I only used two sheets for the S-2020-P, but I got 100% success. I just set the iron to its hottest setting — there is some range of temperatures specified to work, but goodness knows how to set a specific temperature with the iron I have.

The component side of the boards needs to be labelled so that jumper wires and components can be inserted correctly. I did this previously by printing on to overhead transparency sheets and attaching these to the boards. This time, I tried laser printable sticky backed transparent sheets which I found on E-Bay sold by **'madaboutink'**. I checked that the adhesive was not conductive

Figure 20: Items used to etch a board

and they seem to work well (but cost about £1 per sheet, so they are not cheap).

Figure 19 shows the *Press-N-Peel* 'paper' and what it looks like after the design has been transferred to the copper of the board with a hot iron. Some of the tracks near the edge did not adhere and can be seen as black lines (this is the toner deposited by the laser printer). It is possible to patch up small problems by filling in missing track sections on the copper with red Staedtler LumoColor 313-2 pens before etching, as suggested by Sandra Noble Goss: http://gossdesignstudio.com/sandranoblegoss/etching.html — this red colour works well while some other colours do not.

All the unreliability of home made PCBs lies in transferring the design to the copper, in my experience. Etching (with traditional ferric chloride) has never failed. This is with either premixed ferric chloride or stuff made up from dry granules. I use the simplest method: I put the board in a food container, add ferric chloride and repetitively tip the container back and forth so that the etchant flows over the board — just keep going until the copper has gone in the places it needs to go! If the resist (whatever it is) is sound (the design has been correctly transferred by whatever means), the copper in the track areas will be fine. Sometimes (in winter) I put the food container in a cat litter tray filled with hottish water, which keeps etching times reasonable (less than 20 minutes).

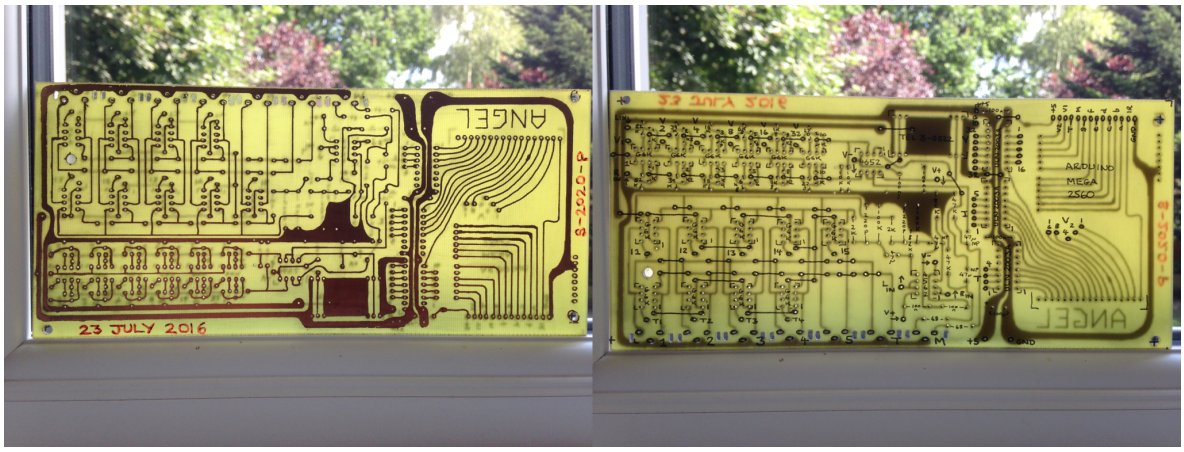


Figure 21: The etched board, drilled and with the component side film attached

Figure 20 shows the items needed to etch the board. All very basic. After etching, the toner covering the tracks can be removed with acetone. The results are shown in Figure 21 — after drilling and having the component side adhesive film attached.

Drilling the holes for component leads must be done with a small drill in a small drill press using tungsten carbide micro drills (e.g. from Gloster Tooling on E-Bay UK)[19]. Useful sizes are 0.85 mm or 0.9 mm for most leads and 1.2 mm for chunkier items or for crudely milling out bigger, irregular, shapes. These bits are extremely sharp and very effective — but very brittle. They will randomly break — and it is essential to keep the board absolutely flat when drilling it. A bit may drill a thousand holes or it might break on the second! Without having the small drill mounted in a proper drill press, it is likely that all bits will break on the second hole! Or pretty quickly, anyway. The drill press attachment is vital, in my opinion.

Bigger holes (for screws) are best drilled with a full size drill press — which is also essential for metalwork for cases and so on. The small and full size drill presses I use are shown in Figure 22. I have had more fun with these for the money they cost than almost any other thing I've bought — for some definition of fun. . . The big one is a Clarke Metalworker from Machine Mart that cost about £50 — which after many years of trouble free use, I regard as excellent value. The small one is a Proxxon MicroMot 50/E and the drill press attachment is a Proxxon MB 140/S — those two together were quite expensive: the 50/E with power supply is about £65 and the MB 140/S is about another £50. They too have worked very well for many years, though[20].

D.1 'Interconnects'

To go with the S-2020-P, I replaced nearly all the previous 'interconnects' — i.e. bits of wire with RCA plugs on the end. Some of the ones I had been using were a bit ratty. After cutting one of them and looking at the innards, even I thought it was time to improve things. Another issue is that some of the previous cables were far (far) too long. Cables of pretty much exactly the right length can be made up from some cable and plugs.

[19] HSS bits are not satisfactory in this application.

[20] Although only with occasional use, I suppose. If used 8 hours a day in commercial production, I don't really know how long they would last.

Figure 22: The Little and Large of drill presses

Very decent cables can be made up from RG-58/U coax, which costs 50p per metre. Suitable RCA plugs for use with this cable are Neutrik/Rean NYS373. These cost less than £1.20 each, so a 1 metre cable can be made up for less than £3 (£6 for a stereo pair). Such cables are electrically and physically as good as any that can be bought — at any price[21].

D.2 Drawing Circuit Diagrams with Inkscape

If you use a tool like Kicad to design PCBs, you will have a nicely drawn circuit diagram produced as part of the process. Doing things 'by hand', however, means you don't automatically have a pretty circuit diagram. I dream of software which could automatically turn a hand drawn diagram into a 'publication quality' CAD drawing — but this is just a dream![22]

Inkscape is a very capable open source vector graphics drawing program that in recent versions (from V0.91) has had a symbol library facility. This defines symbols in SVG which can then be used in drawings. Although there are SVG circuit diagram component libraries on the Internet, none of these seem to 'just work' with the Inkscape symbol library facility. On the other hand, there are SVG symbol libraries which do work (and ship) with Inkscape — but not for circuit diagrams. Reluctantly, I decided to make a suitable library myself.

[21] Of course, some will *never* agree with this — especially those making *huge* profits from manufacturing and selling 'interconnects'.

[22] I am surprised that there really seems to be nothing at all like this that I can find. Perhaps I haven't looked hard enough (there was an article about MIT having something a few years back that did something like this, but it seems to have come to nothing). Maybe it is just too hard — it definitely wouldn't be easy.

This was done by plotting the circuit symbols out on squared paper then using the coordinates in a Python program that writes SVG using the **svgwrite** package. This can be installed by:

```
sudo pip install svgwrite
```

The program I wrote with this to generate the symbol library is called `makesymbols.py`. When run, it generates a file: `mycircuits.svg` (!) which is the SVG symbol library.

The library uses a mixture of symbol styles. They happen to be the ones I grew up with, and it doesn't matter to me that they don't conform to one set of rules — but that might be a problem in other contexts. There are a limited number of symbols — just the ones I have needed so far. However, they are enough for many diagrams and it would be easy (if tedious) to add more component types (there are no FETs at the moment, for instance). A title is put in the library so it appears as 'Electronics Symbols' in the Inkscape Symbols window.

To use the library, put `mycircuits.svg` in: `~/.config/inkscape/symbols` That directory may not exist. Just create it if not. The library should then be visible the next time Inkscape is started.

Here are some settings I have found useful when drawing circuit diagrams in Inkscape:

- Document properties settings:
  - For largish diagrams, use A1 paper in Landscape. Paper size isn't that important.
  - Set up a grid: Units: px; Spacing: 5 in x and y; Major grid line: 5; enabled; visible; snap to visible only (or maybe not?).
  - Snap at defaults (I think).
- Set default stroke width with pen to 2 by:
  - Draw a pen stroke.
  - Set its width to 2.
  - Make sure it is selected.
  - Go to Preferences:
    - Pen section:
    - This tools own style: Yes
    - Take from selection (press it).
- In main UI:
  - Snap cusp nodes: Yes

The S-2020-P circuit diagram in Figure 9 was drawn in Inkscape with this symbol library and came out quite well, I think. It would still be great if someone came up with a 'sketch to lovely diagram' program though, because it does take some effort with Inkscape.

E S-2020-P Parts List

This list is not complete, but it does give Farnell UK part numbers (mainly) and minimum purchase quantities for most items.

Omron G6K-2P 5DC relays	4446082	15 off
Pro-Signal PSG01550 dual RCA sockets	1280669	7 off
TruOpto OSL-10561-LR red seven segment display TruOpto OSL-10561-LG green seven segment display	57-0121 (Rapid) 57-0131 (Rapid)	1 off 3 off
LM4562 dual op-amp 8 pin DIL	1685366	2 off
Vishay TSOP 4838 38 kHz IR receiver	4913190	1 off
Traco Power TEL 3-0522 5V to +/-12V DC-DC converter	1204953	1 off
68Ω 0.6W metal film resistor, MRS25 series 270Ω 0.6W metal film resistor, MRS25 series 470Ω 0.6W metal film resistor, MRS25 series 1KΩ 0.6W metal film resistor, MRS25 series 1K8Ω 0.6W metal film resistor, MRS25 series 2KΩ 0.6W metal film resistor, MRS25 series 2K2Ω 0.6W metal film resistor, MRS25 series 3K9Ω 0.6W metal film resistor, MRS25 series 5K6Ω 0.6W metal film resistor, MRS25 series 6K8Ω 0.6W metal film resistor, MRS25 series 8K2Ω 0.6W metal film resistor, MRS25 series 10KΩ 0.6W metal film resistor, MRS25 series 18KΩ 0.6W metal film resistor, MRS25 series 39KΩ 0.6W metal film resistor, MRS25 series 82KΩ 0.6W metal film resistor, MRS25 series 100KΩ 0.6W metal film resistor, MRS25 series	 9466487 9468463 9465170 9465448 9466711 9467912 9469230 9469800 9470441 9463976 9465030 9467718 9470271	1 off (10) 1 off (10) 1 off (10) 1 off (10) 1 off (10) 1 off (10) 1 off (10) 1 off (10) 1 off (10) 1 off (10) 1 off (10) 1 off (10) 1 off (10) 1 off (10) 1 off (10) 1 off (10)
220pF polypropylene capacitor 10% 100nF polypropylene capacitor 5% 47μF non-polarized electrolytic capacitor	1890206 1890277 use 2.2 μF PP instead	1 off (10) 1 off (10) 4 off
ULN2003APG Darlington relay drivers	E-Bay	3 off
220 mm x 100 mm single sided PCB	Spiratronics	2 off
Arduino Mega 2560 (Sainsmart)	E-Bay	1 off
GX16–2 chassis plug GX16–2 cable socket USB "B" plug and cable	E-Bay E-Bay E-Bay	1 off 1 off 1 off

F S-2020-R Remote Controller

I just couldn't help myself in the end . . . I had to try to make a remote control for the S-2020-P. Looking around on the Internet, people had managed to get low power consumption systems built around Arduino Pro Mini boards. They seemed to be using them for data logging where they were woken up rarely from a deep sleep mode. This seemed like a possible way to go.

Unfortunately, it turned out not to be. I needed to use Ken Shirriff's IR library to transmit codes

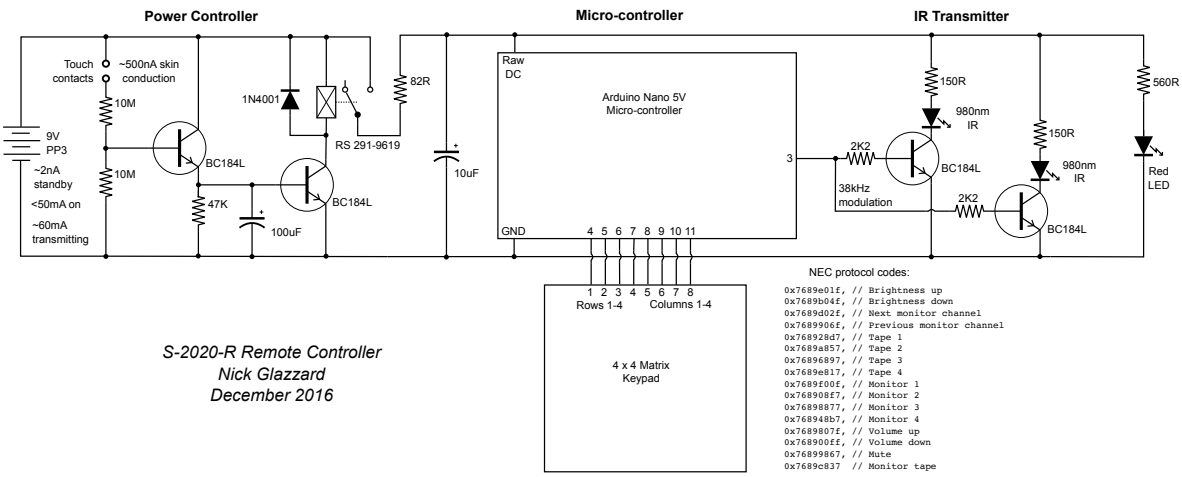


Figure 23: Remote Control Circuit

and a library for scanning a matrix keyboard. I couldn't see any way of using those with the sleep mode library. The problem was how to wake up when some event happened. It would probably be possible to do this by modifying the Keypad library so that a wakeup was generated when any key was pressed . . . but I'm not sure of this as I ended up using a different (much more hacky) approach.

If you have a system that uses 20mA or more and you want it to run for a long time on a battery, you have two choices (given that making the system sleep most of the time is not an option):

- Use a (very) big battery. Some sort of rechargeable thing would be needed. Possible — but the device would have to be recharged nightly in all likelihood (just as tablets, phones and so on do). That wasn't very appealing.
- Only turn it on when you actually want to use it. Having a power switch on a remote (that you need to remember to turn off) is pretty lame. Or is it? To use a remote you have to touch it. If it could be arranged that touching the thing turned it on and letting go of it turned it off, it might not be lame after all.

So, the solution I followed was to design a mechanical package that sensed when it was being held (or touched) and turned on the electronics when in contact with skin. It ended up being built with an Arduino Nano — only because I bought a Mini Pro but was accidentally sent a Nano!

The circuit diagram for the thing is shown in Figure 23. Current consumption is minuscule until the device is touched at which point about 500nA passes through the person touching it and the system comes to life[23]. Then it uses several tens of milli-amps . . . but typically only for a few seconds.

As very much a prototype, the thing was assembled on strip board and put in a custom built case made from aluminium and acrylic plastic. Bridging the two aluminium sides with skin (a hand normally!) will turn the device on. The object is shown in Figure 24.

[23]The power controller part of the circuit is a modification of that in a paper by Burton Slotnick: *A Simple 2 Transistor Touch or Lick Detector Circuit*, J. Exp. Anal. Behav. (seriously!), March 2009

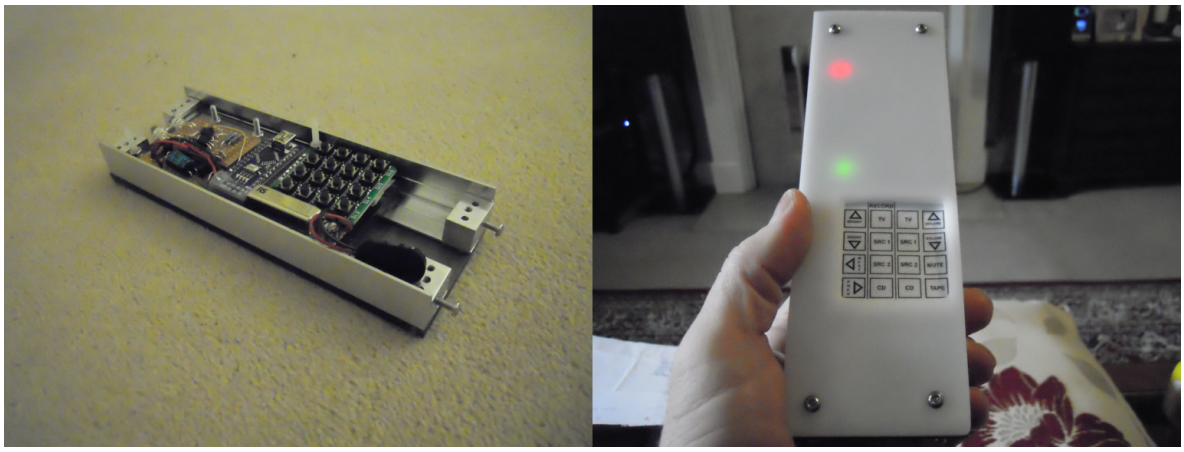


Figure 24: Finished Remote Control

It actually works surprisingly well and cost almost nothing. All components came from the spare parts bin except for the Arduino Nano (about £3), the 980nm IR LEDS (about 10p each), and the matrix keypad (also about £3 from E-Bay). It is powered by a PP3 alkaline battery (around 500maH is likely). The keypad is (it turns out) a readily available item — how durable it is remains to be seen, but the key action is quite pleasantly positive. The keypad is covered by a flexible membrane on which the key functions have been printed. Said 'flexible membrane' is nothing more than a piece of paper covered both sides with sticky backed transparent plastic! The annotations were done in Inkscape and printed on a laser printer. This may or may not last long — but it is easily replaced when it wears out!

The firmware is very simple indeed, and is given below:

```
#include <Keypad.h>
#include <IRremote.h>

// Keypad scanner object.
const byte ROWS = 4;
const byte COLS = 4;

char keys[ROWS][COLS] =
  {
    {'+','A','1','U'},
    {'-','B','2','D'},
    {'N','C','3','M'},
    {'P','X','4','T'}
  };
byte rowPins[ROWS] = {4,5,6,7};
byte colPins[COLS] = {8,9,10,11};
Keypad kpd = Keypad(makeKeymap(keys),rowPins,colPins,ROWS,COLS);

// IR remote sender object.
IRsend irsend;

void setup()
{
  Serial.begin(9600);
}

void loop()
{
  static char keyin[] =
    { '+','-','N','P',
      'A','B','C','X',
      '1','2','3','4',
      'U','D','M','T'};
  static unsigned long codeout[] =
    { 0x7689e01f, // Brightness up
      0x7689b04f, // Brightness down
      0x7689d02f, // Next monitor channel
      0x7689906f, // Previous monitor channel
      0x768928d7, // Tape 1
```

```
      0x7689a857, // Tape 2
      0x76896897, // Tape 3
      0x7689e817, // Tape 4
      0x7689f00f, // Monitor 1
      0x768908f7, // Monitor 2
      0x76898877, // Monitor 3
      0x768948b7, // Monitor 4
      0x7689807f, // Volume up
      0x768900ff, // Volume down
      0x76899867, // Mute
      0x7689c837  // Monitor tape
    };
  char key = kpd.getKey();
  if( key ){
    for( int i=0; i<16; i++ ){
      if( key == keyin[i] ){
        Serial.print(key); Serial.print(codeout[i],DEC); Serial.println();
        irsend.sendNEC(codeout[i],32);
        delay(100);
      }
    }
  }
}
```

I will be the first to admit that this is not a ‘professional’ solution to providing a remote control. It is, however, effective (amazingly) and its quirkiness is not entirely unattractive (to me, anyway).

One improvement would be to overwrite the boot loader on the Arduino, as this adds about a 2 second delay to turn-on (which would otherwise be nearly instant). This could be done with an ISP programmer (built from a second Arduino Nano) . . . but I haven’t done it yet.

G Acknowledgements

The devices described here are partially or entirely based on many other people's work. This is especially true of the S-2020-S, which is just an assembly of freely available software components and fully assembled hardware modules. Thanks are due to the following:

- The piCorePlayer team responsible for — guess what? https://sites.google.com/site/picoreplayer/home/news
- The Tiny Core Linux team for the version of Linux used by piCorePlayer. http://distro.ibiblio.org/tinycorelinux
- The Squeezelite team for the software Squeezebox emulator used by piCorePlayer. https://code.google.com/archive/p/squeezelite
- Slim Devices and Logitech for creating Logitech Media Server and making it open source. http://www.mysqueezebox.com/download
- 'Triode' for their Spotify and BBC iPlayer plug-ins for Logitech Media Server.
- The Raspberry Pi Foundation for the Raspberry Pi embedded computer. https://www.raspberrypi.org
- IQaudIO Limited for producing very high quality DACs for use with Raspberry Pi devices. http://www.iqaudio.co.uk
- The creators of the Arduino micro-controller platform. https://www.arduino.cc
- Jos van Eijndhoven for providing a design resource for logarithmic attenuators and describing his pre-amplifiers based on them.
- Mark Hennessy for describing his pre-amplifier.
- Ken Shirriff for his infra-red remote library for the Arduino.
- Dean Reading for his seven segment display library for the Arduino.
- Alexander Brevig and Mark Stanley for their Keypad library for the Arduino.
- The huge number of people responsible for the success of the GNU/Linux operating system. This created the environment in which these kinds of computer based DIY projects are feasible. Without it (or something very like it), these sorts of things would be quite inconceivable.