YoloDROID • GlusterFS • Cloudshell • Stereo Boom Bonnet • Boombox

ODROID Magazine

Year Four
Issue #48
Dec 2017

Hands free:

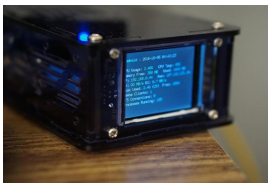
BLUETOOTH A2DP AND HFP UNIT WITH ODROID-XU4

HOME ASSISTANT : USING INFRARED, MOTORS, AND RELAYS

Using your ODROID-XU4 as a Bluetooth A2DP Speaker or HFP Handsfree Unit With Your iPhone

December 1, 2017

The first is to stream audio via Bluetooth A2DP from an iPhone to an ODROID-XU4, and the second is to use the ODROID-XU4 as an HFP Handsfree Unit for the iPhone during calls.



Custom Status Display For The ODROID CloudShell and CloudShell 2

December 1, 2017

This article is not written to be a step-by-step guide on creating custom status display, but instead focuses on a general approach.

Stereo Boom Bonnet: A Great Way To Enjoy Music On Your ODROID

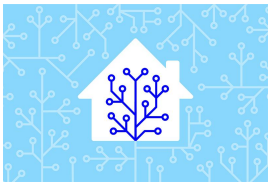
December 1, 2017

The I2S 2Watt Stereo Boom Bonnet Kit (https://goo.gl/1mXXVH) is a compact speaker system for the ODROID-XU4 and ODROID-C1+/C2. It uses I2S as digital sound standard for audio output. It is very easy to install, and you'll be rockin' out in 15 minutes. To connect it to your ODROID, follow these

Boom Box: Sound Engineering a Better Speaker

December 1, 2017

To improve the sound of Hardkernel's Stereo Boom Bonnet (https://goo.gl/TrDU8u), I went to my local hobby shop and got an "assortment pack" of styrene which happened to have some .080mil sheets and some tubing in it. I also purchased a little styrene solvent for welding. I did everything by eye,



Home Assistant: Using Infrared, Motors, and Relays

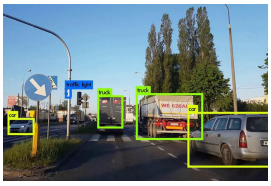
December 1, 2017

In this article, we are going to combine ODROIDs with a little bit of hardware and Home Assistant, so that we can start converting non-smart appliances to "smart" appliances. We'll be working with wires, doing a bit of soldering, connecting relays, and in some cases working with mains voltage, so

Android Gaming: Stranger Things, Pocket Morty, and Streets of Rage

December 1, 2017

It's easy to chill with your ODROID because there are so many Android gaming options for the Android operating system. In this article, we'll explore three new releases: Stranger Things: The Game, Pocket Morty's, and Streets of Rage. Stranger Things: The Game Straight from the hit TV show, here comes



Running YOLO On ODROID: YOLODROID

December 1, 2017

This guide tells you how to get TinyYOLO installed and running on your ODROID-XU4.

Linux Gaming: Need for Speed II Second Edition

December 1, 2017

Need for Speed II Second Edition (NFS2SE) was released with 3DFX support, more tracks and cars, along with mirror mode and backward track mode, making it quite an improvement over the original Need for Speed II.

Exploring Software-Defined Storage with GlusterFS on the ODROID-HC1: Part 2 – Client Performance

December 1, 2017

I am going to show you how to setup NFS and Samba clients to access the GlusterFS volume and compare the performance of the different clients.

Meet An ODROIDian: Andrea Cole, Assistant Editor of ODROID Magazine

December 1, 2017

Please tell us a little about yourself. I'm currently a sales admin for Lab Manager, an industry-focused publication for the scientific community. I've been a part of their parent company, LabX Media Group, for over 10 years, and have been working specifically in the Lab Manager division for four years. ▶️

Using your ODROID-XU4 as a Bluetooth A2DP Speaker or HFP Handsfree Unit With Your iPhone

December 1, 2017 By Dennis Chang ODROID-XU4, Tinkering

Two use cases are presented in this article that are of special interest to those building car computers using ODROIDs. The first is to stream audio via Bluetooth A2DP from an iPhone to an ODROID-XU4, and the second is to use the ODROID-XU4 as an HFP Handsfree Unit for the iPhone during calls. We will be using the Bluez 5 bluetooth stack, the oFono mobile application development framework and the PulseAudio sound system software for this project. Although I have not tested it, this procedure should work with an Android phone.

A2DP is very easy to setup and is completed on the way to setting up handsfree. HFP is more difficult to setup because it requires building PulseAudio 11 from source and replacing the PulseAudio 8 package preinstalled in Ubuntu MATE. I have divided this article into two sections so that those of you only interested in A2DP can avoid the more difficult steps in getting HFP to work. If you are only interested in the Bluetooth speaker functionality, you can use the ODROID-XU4's built-in audio out over the HDMI port instead of the C-Media CM108-based USB audio adapter. To keep things simple, this article assumes you will be using the USB audio adapter listed below.

Before you start, note that documentation and discussions for the latest versions of Bluez 5, oFono, and PulseAudio are somewhat sparse. Most of everything I found online for this project's use case was too old, discussing prior versions of each software. If you run across a problem trying out this project, there is a fairly high chance you will not find anyone knowledgeable enough to help, myself included! I advise you to not stray too far from these instructions and hardware selection until you have gotten your setup working properly, then you can get creative knowing you have a working baseline to go back to.

Phase 1: A2DP

A2DP can be accomplished using the packages preinstalled with Ubuntu 16.04.3 MATE: Bluez 5.37, oFono 1.17, and PulseAudio 8. Start with an ODROID-XU4 running the official Ubuntu 16.04.3 with 4.9 kernel MATE image and run all the OS updates before starting this project. I used the ubuntu-16.04.3-4.9-mate-odroid-xu4-20170824.img image file. You are probably better off choosing the ODROID-XU4 instead of the ODROID-XU4Q, since the extra CPU performance may help reduce call audio latency or improve audio quality if you decide to go further and optimize PulseAudio's resampling.

The specific devices I used for this project are:

- iPhone 5S
- Cambridge Silicon Radio Bluetooth USB adapter: http://bit.ly/2gNybJW.
- Sanwu Audio SW-HF07 USB audio adapter (with a C-Media CM108 chipset that is known to work on ARM LINUXes with the built-in driver): http://bit.ly/2zEBIWZ.
- Headset with separate 3.5mm microphone and headphone plugs for testing

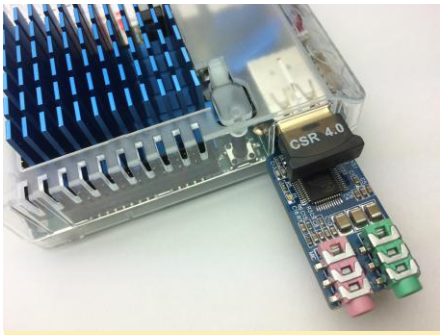


Figure 1 – Closeup of the USB bluetooth and audio adapters

Insert the USB Bluetooth and audio adapters into available USB ports on the ODROID-XU4, then insert the headset's headphone and microphone plugs into the corresponding jacks on the audio USB adapter.

Selecting the correct audio interface

Log in to the MATE desktop as the default user "odroid". This step is important because PulseAudio is setup to run in per-user mode by default, so it will start automatically after you log in but is not running when the logon screen is being displayed. We will not be setting up PulseAudio to run in system-wide mode in this article, as it introduces extra challenges.

Test the audio using the built-in Sound Preferences application in MATE. Change and save the configuration as necessary, testing the sound as needed. Since I am using a C-Media USB audio adapter, I had to select it as the default Input and Output device as opposed to the ODROID-XU4's

built-in audio (output through the HDMI port). Leave the Sound Preferences application open so that PulseAudio is running in your user session.

Figure 2 – Selecting the USB Audio Device as input in the Sound Preferences control panel

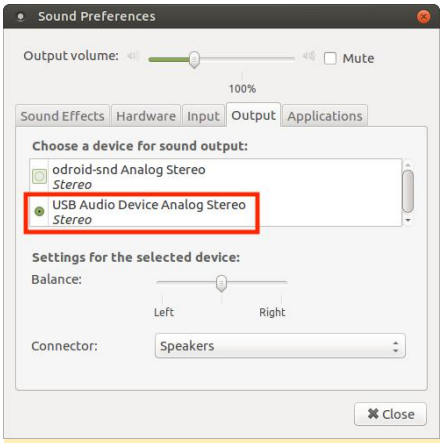


Figure 3 – Selecting the USB Audio Device as output in the Sound Preferences control panel

In the /home/odroid/.config/pulse directory, look for the files ending in "-default-sink" and "-default-source" and write down the filenames and their contents. You should see something like this:

```
dc87f36fc06c441a85ff7269baabcdef-default-sink:
 alsa_output.usb-C-Media_Electronics_Inc._USB_Audio_Device-00.analog-stereo

dc87f36fc06c441a85ff7269baabcdef-default-source:
 alsa_input.usb-C-Media_Electronics_Inc._USB_Audio_Device-00.analog-mono
```

You want the selected device in both files to be the audio interface that has the microphone, which is typically not the built-in HDMI audio output. Use the Sound application to test the audio input and output before moving on to the next step.

Bluetooth pairing with your iPhone

Bluez 5.x is already preinstalled in the Ubuntu 16.04.3 MATE image, so all we have to do is pair your iPhone to the ODROID-XU4. We will use the included bluetoothctl command, which must be run with root privileges, or it will error out:

```
$ sudo -s
# bluetoothctl
 [bluetooth]# show
```

You should see something like this:

```
Controller 00:AA:BB:CC:DD:11
 Name: odroid
 Alias: odroid
 Class: 0x1c0000
 Powered: yes
 Discoverable: no
 Pairable: yes
 UUID: Headset AG (00001112-0000-1000-8000-###########)
 UUID: Generic Attribute Profile (00001801-0000-1000-8000-###########)
 UUID: A/V Remote Control (0000110e-0000-1000-8000-###########)
 UUID: OBEX File Transfer (00001106-0000-1000-8000-###########)
 UUID: Generic Access Profile (00001800-0000-1000-8000-###########)
 UUID: OBEX Object Push (00001105-0000-1000-8000-###########)
 UUID: PnP Information (00001200-0000-1000-8000-###########)
 UUID: A/V Remote Control Target (0000110c-0000-1000-8000-###########)
 UUID: IrMC Sync (00001104-0000-1000-8000-###########)
 UUID: Audio Sink (0000110b-0000-1000-8000-###########)
 UUID: Audio Source (0000110a-0000-1000-8000-###########)
 UUID: Vendor specific (00005005-0000-1000-8000-###########)
 UUID: Message Notification Se.. (00001133-0000-1000-8000-###########)
 UUID: Phonebook Access Server (0000112f-0000-1000-8000-###########)
 UUID: Message Access Server (00001132-0000-1000-8000-###########)
 Modalias: usb:v1D6Bp0246d0525
 Discovering: no
```

If PulseAudio was not running, the list of profiles would be much shorter.

If "Powered" is not "yes", type the following command:

```
# power on
```

Now, let us start pairing by scanning for nearby devices:

```
# scan on
```

On your iPhone, go to Settings > Bluetooth and make sure it is turned on and discoverable. You will eventually see your iPhone show up in the scan by its Bluetooth MAC address. Write this MAC address down as you will use it repeatedly in place of [MAC] below.

```
# scan off
# agent KeyboardOnly
# default-agent
# pair [MAC]
```

This might fail; try again until it succeeds in initiating the pairing and asks for the passkey. Look on your iPhone for the passkey and input it when it says:

```
Attempting to pair with [MAC]
 [CHG] Device [MAC] Connected: yes
 Request passkey
 [agent] Enter passkey (number in 0-999999): ######

# connect [MAC]
# trust [MAC]
# info
```

You should see details on your iPhone and its Bluetooth profiles. On your iPhone, it should show that the device called "odroid" is connected.

```
# exit
```

Figure 4 – Connecting the ODROID in the iPhone Bluetooth settings

At this point, you should be able to send audio playback from your iPhone's iTunes app to the ODROID-XU4 over Bluetooth. If you cannot, you should go into the iPhone's Bluetooth preferences and force the reconnect to "odroid," even if it is already connected.

Figure 5 – Sending audio playback via Bluetooth from iTunes to the ODROID-XU4

It is also possible to hear the dialing when using the Phone app, but because we have not yet set up the ODROID-XU4 with HFP, it will not receive the audio once the Phone app connects the call.

Phase 2: HFP

The next step is to install Ofono 1.17.x for the handsfree Bluetooth profile, since it is not preinstalled with the OS image. Assuming that we are still in the same "sudo -s" session, type the following command:

```
# apt-get install ofono
```

Check to see that the Bluetooth profile has been added:

```
# bluetoothctl
 [bluetooth]# show

Controller 00:AA:BB:CC:DD:11
 Name: odroid
 Alias: odroid
 Class: 0x3c0000
 Powered: yes
 Discoverable: no
 Pairable: yes
 UUID: Headset AG (00001112-0000-1000-8000-
############)
 UUID: Generic Attribute Profile (00001801-
0000-1000-8000-############)
 UUID: A/V Remote Control (0000110e-0000-
1000-8000-############)
 UUID: OBEX File Transfer (00001106-0000-
1000-8000-############)
 UUID: Generic Access Profile (00001800-
0000-1000-8000-############)
 UUID: OBEX Object Push (00001105-0000-1000-
8000-############)
 UUID: PnP Information (00001200-0000-1000-
8000-############)
 UUID: A/V Remote Control Target (0000110c-
0000-1000-8000-############)
 UUID: IrMC Sync (00001104-0000-1000-8000-
############)
 UUID: Audio Sink (0000110b-0000-1000-8000-
############)
 UUID: Audio Source (0000110a-0000-1000-
8000-############)
 UUID: Handsfree (0000111e-0000-1000-8000-
############)
 UUID: Vendor specific (00005005-0000-1000-
8000-############)
 UUID: Message Notification Se.. (00001133-
0000-1000-8000-############)
 UUID: Phonebook Access Server (0000112f-
0000-1000-8000-############)
 UUID: Message Access Server (00001132-0000-
1000-8000-############)
 Modalias: usb:v1D6Bp0246d0525
 Discovering: no
```

Note that we now have the Handsfree profile (5th from the bottom of the UUID section of the list). Next, quit bluetoothctl:

```
# exit
```

At this point, it is possible to start a phone call with the iPhone Phone app and select the "odroid" device as the handsfree audio, and you will hear the keypad tones as you dial the call, but as soon as the call starts, the ODROID-XU4 will drop the audio and cause the iPhone to switch from "odroid" to its internal speaker and internal microphone.

Here is where a little experimentation is required. The Ubuntu-packaged PulseAudio 8 apparently has a bug or is missing a feature that causes the call audio to drop and fills /var/log/syslog with these error messages (visible if you set PulseAudio to debug logging):

```
D: [bluetooth] module-loopback.c: Requesting
rewind due to end of underrun.
I: [alsa-sink-bcm2835 ALSA] module-
loopback.c: Could not peek into queue
```

The solution is to uninstall PulseAudio 8 and then build and install PulseAudio 11.1 (the latest version as of this writing) from source code, as detailed below.

You should still be in the same "sudo -s" session after running bluetoothctl. If not, type the following command:

```
$ sudo -s
```

You might want to make a backup of the PulseAudio autostart file here to be reused later:

```
# cp /etc/xdg/autostart/pulseaudio.desktop ~
# apt-get remove pulseaudio
# apt-get autoremove
# dpkg --purge pulseaudio
```

It might be a good idea to remove the old PulseAudio config folder:

```
# rm -fr /etc/pulse
```

PulseAudio 8 has now been removed, so now we get and build and install PulseAudio 11.1:

```
# apt-get build-dep pulseaudio
# apt-get install git
# exit
$ cd ~
```

Get the source code with git using one of the two commands below:

```
$ git clone
git://anongit.freedesktop.org/pulseaudio/pul
seaudio
```

or:

```
$ git clone
http://anongit.freedesktop.org/git/pulseaudi
o/pulseaudio.git
```

PulseAudio is also released in compressed archives if you do not want the development version in the git repository.

```
$ cd pulseaudio
$ export CFLAGS=-fomit-frame-pointer
$ ./autogen.sh
$ make
```

The build will take about 15 minutes and will issue a lot of warnings, but should end without any major errors. If the build went well, and it should, you may install PulseAudio.

```
$ sudo make install
```

Note that PulseAudio built from source places its config files in /usr/local/etc/pulse, not in /etc/pulse as the Ubuntu-provided PulseAudio does.

There is one setting we will enable in PulseAudio's config file to allow the mono microphone to be remixed to stereo. If not done, the microphone audio will be discarded.

```
$ sudo vi /usr/local/etc/pulse/daemon.conf
```

Uncomment the following line by deleting the semicolon in front of it and save the file:

```
enable-remixing = yes
```

You must reboot the ODROID-XU4 and log back in as the odroid user now, otherwise PulseAudio will misbehave and cause crackling on the microphone audio. Rebooting will cause the iPhone to lose its Bluetooth connection to the ODROID. Next, you can start PulseAudio (without the benefit of the scripts that automatically started it on logon):

```
# pulseaudio --start -D
```

At this point, you should be able to route the call audio to the ODROID. Reconnect your iPhone to the "odroid" device by going to Settings > Bluetooth and tapping "odroid" in the list of paired devices.

Start by playing some music using the iPhone's iTunes. Then, place a call and hear the audio from the call through the ODROID. The final test is to speak through the microphone and have the other side confirm they can hear you.

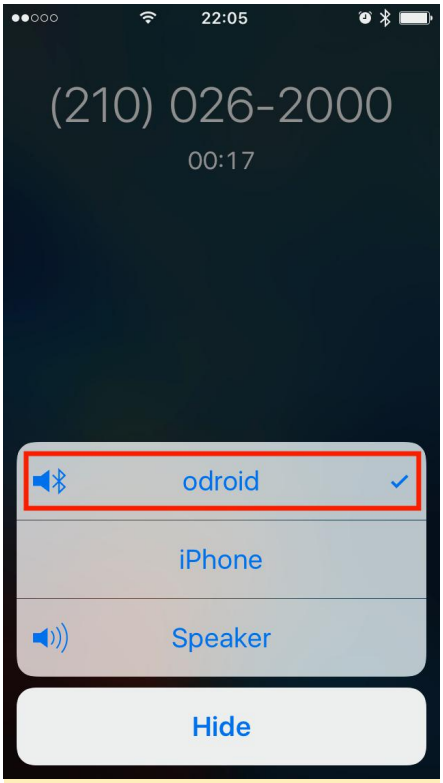


Figure 6 – Testing the microphone during a phone call using the iPhone

Finishing up

Note that some of the features of the Ubuntu-provided Sound panel are now broken because we built and installed PulseAudio from scratch. Notably, the "Test Speakers" utility on the Hardware tab does not seem to work any more, and you will not hear MATE UI sound effects. Yet, I was still able to visually monitor the microphone input on the Input tab and adjust the volume using the Volume slider in the system tray. Setup PulseAudio to start automatically in per-user mode by creating this file (or copy the old one back in place, a process I did not test):

```
/etc/xdg/autostart/pulseaudio.desktop

[Desktop Entry]
 Version=1.0
 Name=PulseAudio Sound System
 Comment=Start the PulseAudio Sound System
 Exec=start-pulseaudio-x11
 Terminal=false
 Type=Application
 Categories=
 GenericName=
 X-GNOME-Autostart-Phase=Initialization
 X-KDE-autostart-phase=1
 NoDisplay=true
```

Reboot the ODROID-XU4 and log in as the "odroid" user, then make sure PulseAudio is running before reconnecting your paired iPhone.

Troubleshooting

The above procedure has been tested carefully several times, so I did not cover a lot of troubleshooting here. If

something does go wrong, I have found that these simple steps often work to set things right:

1) Reboot the ODROID-XU4 (remembering to start PulseAudio if you are not starting it automatically)

2) Go into iOS Settings > Bluetooth and make sure the “odroid” device is connected. If it is, then disconnecting it and reconnecting is a good idea.

3) In the worst case, having the iPhone forget the device and the redoing the pairing almost always fixes things.

For any other errors, your best bet is to watch the logs with the following command:

```
$ sudo tail -f /var/log/syslog | grep
'bluetooth\|ofono\|pulse'
```

Bluez 5, oFono, and PulseAudio are each very complex, so you should focus on searching for forum discussions about exactly the error messages you see in the log, if you get any. It is possible to configure each one to increase their logging level. I have found that changing log-level = debug in PulseAudio’s /usr/local/etc/pulse/daemon.conf is enough. It was usually not necessary for me to watch oFono or Bluez debug logs while troubleshooting.

Keep in mind that the audio input and output volume levels are not controlled by the iPhone. They are controlled by PulseAudio, and the easiest controls to use are the volume setting slider in Ubuntu MATE’s system tray and the input slider in the Sound application. Also, some USB audio adapters and wired microphones have very low input gain. Even with the PulseAudio input volume cranked all the way up, they might be too quiet for your needs. You can always try a different microphone, different USB audio adapter, or use a small analog amplifier in between the mic and the USB audio adapter. If you are unhappy with the audio quality, it is possible to tweak PulseAudio to improve its performance and quality. That is a vast subject area and is not covered here.

Finally, I noticed that the iPhone does not aggressively reconnect to the ODROID-XU4 via Bluetooth like it does with my off-the-shelf Bluetooth speaker, which is also Cambridge Silicon Radio chipset-based. There are probably some settings within Bluez that can change the way it connects to paired devices. That is also not covered here.

Conclusion

If you follow the instructions above and stick as closely to the hardware selection as possible, you will have your ODROID-XU4 working properly as an A2DP Bluetooth speaker or HFP Handsfree unit for your car computer project. You are encouraged to build on top of this foundation, such as writing or porting an app that uses oFono’s comprehensive API to control the iPhone via a touchscreen UI on your car computer. If you do figure this next project out, please share your findings with the ODROID community by writing about it for this magazine.

Custom Status Display For The ODROID CloudShell and CloudShell 2

December 1, 2017 By Mike Partin CloudShell, ODROID-XU4

This article is not written to be a step-by-step guide on creating custom status display, but instead focuses on a general approach. To follow along, you will need a basic level of programming knowledge. Familiarity with any programming language will do, as I will look into where and how to find the information we need for our goal. That being said, I also included is a link to the project I wrote along this article, which is written in the Go language.

If you have a ODROID CloudShell or CloudShell 2 case, you most likely have been using the cloudshell-lcd package, and have seen how useful that information display is. My problem came from wanting more information to be displayed, and from the small text being hard to read on the CloudShell from across the room. I wanted something more visual that could be read quickly and instantly understood. Progress bars, seemed like a good enough answer, so there I went. Most of the information necessary can be gathered by reading files in /proc on Linux, this could be accomplished with the linproc filesystem.

CPU Usage

Knowing that I wanted CPU, RAM, swap, networking, and disk statistics, I had a good place to start. Linux has made CPU usage, and most other stats, pretty easy as the information available in '/proc/stat'. More information is available at **http://bit.ly/2jGKrRd**. The first line gives us an aggregation of all the core statistics, with the following fields represented:

* user: Time spent in user mode
* nice: Time spent in user mode with low priority (nice)
* system: Time spent in system mode
* idle: Time spent in the idle task.
* iowait: Time waiting for I/O to complete. However, unreliable see the proc(5) man page for details.
* irq: Time servicing interrupts
* softirq: Time servicing softirqs
* steal: Stolen time, time spent in other operating systems, in virtualization workloads
* guest: Time spent running a virtual CPU for guest OS's (virtual workloads)
* guest_nice: Time spent running a niced guest (virtual workloads)

Since space is at a premium on our display, we're only worrying about the first line, since it gives us our total stats. The following command will print out only the first line from '/proc/stat'.

```
$ head -n1 /proc/stat
 cpu 817905 909158 818680 133949276 2463 0
11128 0 0 0
```

To gather our statistics, we need a delta, that mean we need to read the value, wait for a period of time, for instance 1 second, and then read another. The difference between these values tells us how busy the system was for that second. The numbers will be odd, they won't seem like timestamps, and there is a good explanation for that, since they aren't. They're a counter for what's called "jiffies". For finer grained measurements, such as actual processor time spent on each attribute, one would need to find the static value HZ from the kernel. This command should get that value.

```
$ zgrep -i hz /proc/config.gz
```

This value is roughly the number of ticks per second, which is 1000 on most intel compatible platforms, but embedded systems often use 100. For our purposes, we can just get a measurement of process vs work time:

```
$ head -n1 /proc/stat ; sleep 1; head -n1
/proc/stat
 cpu 885034 1050588 935349 152731137 2546 0
12670 0 0 0
 cpu 885039 1050588 935350 152731533 2546 0
12670 0 0 0
$ tot1=$((885034 + 1050588 + 935349 +
152731137 + 2546 + 12670))
$ wrk1=$((935349 + 152731137 + 2546 +
12670))
$ tot2=$((885039 + 1050588 + 935350 +
152731533 + 2546 + 12670))
$ wrk2=$((935350 + 152731533 + 2546 +
12670))
$ tot3=$((${tot2} - ${tot1}))
$ wrk3=$((${wrk2} - ${wrk1}))
$ python -c "print((${wrk3}.0 / ${tot3}.0) *
100.0)"
```

RAM and Swap Usage

RAM statistics can be pulled from multiple sources. For instance, one could read them from '/proc/meminfo'. However, I chose not to because the values are in kilobytes instead of bytes. I chose a direct syscall rather than opening a file with the resulting file processing. Below is a small program written in Go using CGO instead of the syscall package, which I did for simplicity. It uses

"sysconf(3)", which can gather quick memory statistics, for more information, visit http://bit.ly/2jBNXfl.

```
package main

// #include
 import "C"
 import "fmt"

func main() {
 maxRam := int64(C.sysconf(C._SC_PHYS_PAGES)
* C.sysconf(C._SC_PAGE_SIZE))
 freeRam :=
int64(C.sysconf(C._SC_AVPHYS_PAGES) *
C.sysconf(C._SC_PAGE_SIZE))
 usedRam := (maxRam - freeRam)
 ramPercUsed := (float64(usedRam) /
float64(maxRam)) * 100.0
 fmt.Println("total =", maxRam)
 fmt.Println("free =", freeRam)
 fmt.Println("used =", usedRam)
 fmt.Println("(used / total) * 100 =",
ramPercUsed)
 }
```

This method gives you both the amount of used and available memory pages. This, multiplied by the system's page size constant _SC_PAGE_SIZE, gives us the amount of used and available memory. This statistic was easy, and with less "moving parts" than the CPU usage.

Swap statistics, on the other hand, can easily, reliably, and with no additional calculation overhead, be read out of the "/proc/swaps" file. It looks like this:

```
$ cat /proc/swaps
 Filename Type Size Used Priority
 /dev/sda2 partition 8388604 0 -1
 ...
```

This is pretty self explanatory, and the size and used columns are both measured in bytes.

Network Usage

This one can be fun. First, you need to know what network device you want to measure. You can find this list in multiple ways. One way would be to parse "ifconfig -a" or "ip addr" output. This is a bit cumbersome when compared to other methods such as listing the contents of "/sys/class/net/". On my system, this returns "eth0", "lo", and "wlan0". Getting the interface speed is as simple as using the following command:

```
$ cat /sys/class/net/eth0/speed
 1000
```

The next step is to measure our throughput. We will take periodic values like we used for measuring CPU usage. In this article, I'll be focusing on the eth0 interface, and the data I'm after can be found in "/sys/class/net/eth0/statistics/rx_bytes" and "/sys/class/net/eth0/statistics/tx_bytes". They have a format like the following, which easily gives us basic network metrics:

```
$ cat
/sys/class/net/eth0/statistics/rx_bytes
 324429106
```

Disk Usage

There are two kinds of disk usage we might want to measure here: throughput and capacity. The first can be pulled similarly to many of the other statistics we've gathered so far from the "/proc" directory. The file "/proc/diskstats" will contain data similar to the following:

```
8 0 sda 52410 323 1699276 29193 450364
121816 4772512 41466 0 19286 70376
 8 1 sda1 101 0 6594 93 1 0 8 0 0 70 93
 8 2 sda2 46 0 4424 43 0 0 0 0 0 33 43
 8 3 sda3 52238 323 1686170 29020 448708
121816 4772504 41316 0 19113 70040
 8 16 sdb 81 0 4184 33 0 0 0 0 0 16 33
```

The fields are defined in the Linux kernel source tree in the file "Documentation/iostats.txt":

- Major number
- Minor number
- Device name
- Reads completed
- Reads merged
- Sectors read
- Time spent reading (in ms)
- Writes completed
- Writes merged
- Sectors written
- Time spent writing (in ms)
- Iops currently in progress
- Time spent in iops (in ms)
- Weighted time spent in iops (in ms)

It's easy to see how to find the throughput measurements, but what about disk capacity? One way to do it is with a bit of C (or Go, Rust, Nim, Python, Ruby, or whatever else that can interface to C) and the "statfs(2)" syscall. Go has a syscall package, as I mentioned earlier, that I'm going to use again for this particular example:

```
package main

import (
 "fmt"
 "syscall"
 )

func main() {
 stat := &syscall.Statfs_t{}
 syscall.Statfs("/", stat)
 fmt.Println("Total space:", (stat.Blocks *
uint64(stat.Bsize)))
 fmt.Println("Free space :", (stat.Bfree *
uint64(stat.Bsize)))
 }
```

Saving that code to file as "disk.go" and running it will give you:

```
$ go run disk.go
 Total space: 87352057856
 Free space : 35807154176
```

The new LCD status display on the CloudShell

This project was a fun one, since I got to play with several new things like CGO! I hope you've gotten something out of it as well. The project I alluded to in the beginning can be found on GitHub at http://bit.ly/2hEUa6B.

Stereo Boom Bonnet: A Great Way To Enjoy Music On Your ODROID

December 1, 2017 By Justin Lee Tinkering

The I2S 2Watt Stereo Boom Bonnet Kit (https://goo.gl/1mXXVH) is a compact speaker system for the ODROID-XU4 and ODROID-C1+/C2. It uses I2S as digital sound standard for audio output. It is very easy to install, and you'll be rockin' out in 15 minutes. To connect it to your ODROID, follow these steps:

- Plug 2 x 2 Watt 4 ohm stereo speakers to the connector on the Boom Bonnet board and attach them with glue
- Connect the GPIO ribbon cable one side to the boom bonnet board and the other side to ODROID board
- Update the OS to the latest version

You can easily adjust the audio output level with a potentiometer on the board. It will deliver adequate fidelity sound, but may include some low-grade system noise and is not a replacement for a high fidelity speaker system. The package includes:

- Stereo Boom Bonnet board
- 2 x 2W / 4 ohm mini speakers (28mm diameter, 11.5mm thickness)
- 3 x 5mm PCB spacers
- 3 x 5mm screws
- 200mm 7-pin GPIO ribbon cable (200mm) for the ODROID-C1+/C2 or a 12-pin version for the ODROID-XU4

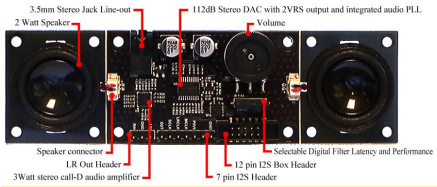


Figure 1 – Annotated diagram of the Stereo Boom Bonnet

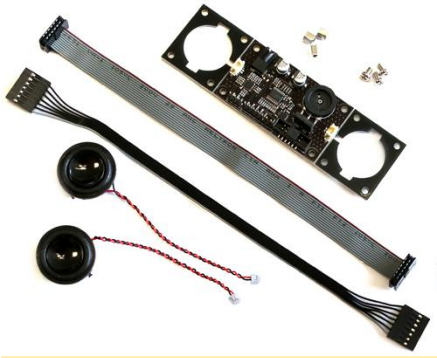


Figure 2 – The Stereo Boom Bonnet includes speakers, spacers, and your choice of ribbon cable

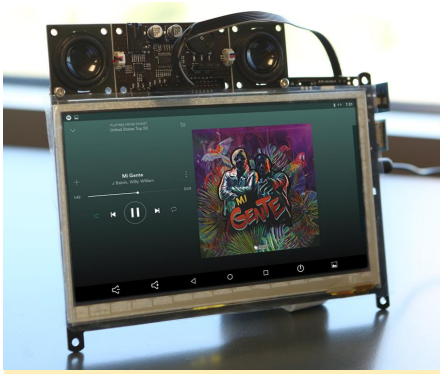


Figure 3 – The Stereo Boom Bonnet easily mounts to the ODROID-VU7

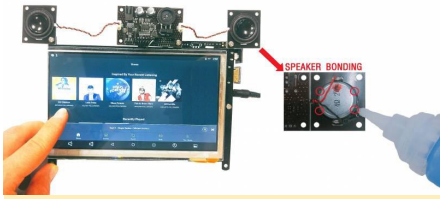


Figure 4 – The Stereo Boom Bonnet speakers can also be separated and attached anywhere with a small amount of glue

Installation

Connect the stereo boom bonnet to ODROID-C1+/C2 using an I2C cable, attach a USB keyboard, USB mouse and HDMI monitor, then power up the system, then update the system:

```
$ sudo apt update && sudo apt dist-upgrade
```

Figure 5 – Attaching the Stereo Boom Bonnet to an ODROID-C1+/C2

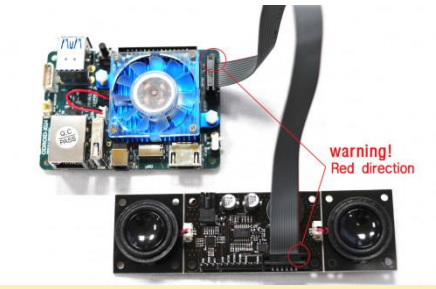


Figure 6 – Attaching the Stereo Boom Bonnet to an ODROID-XU4

Next, ensure that the stereo boom bonnet kernel modules are loaded:

```
$ aplay -l
**** List of PLAYBACK Hardware Devices ****
card 0: ODROIDHDMI [ODROID-HDMI], device 0:
I2S.27 dit-hifi-0 []
  Subdevices: 0/1
  Subdevice #0: subdevice #0
odroid@odroid64:~$
odroid@odroid64:~$ sudo modprobe snd-soc-
pcm5102
odroid@odroid64:~$ sudo modprobe snd-soc-
odroid-dac
odroid@odroid64:~$ aplay -l
**** List of PLAYBACK Hardware Devices ****
card 0: ODROIDHDMI [ODROID-HDMI], device 0:
I2S.27 dit-hifi-0 []
  Subdevices: 1/1
  Subdevice #0: subdevice #0
card 1: ODROIDDAC [ODROID-DAC], device 0:
I2S.27 pcm5102-0 []
  Subdevices: 1/1
  Subdevice #0: subdevice #0
```

Then, navigate to Applications → Sound & Video → Sound → Hardware Tab & Output Tab and select “ODROID-DAC”. If you have to load the driver every time whenever your ODROID-C1+/C2 starts, you can register the driver into /etc/modules and reboot:

```
$ su
Password:  (root password is "odroid")
# echo "snd-soc-pcm5102" >> /etc/modules
# echo "snd-soc-odroid-dac" >> /etc/modules
# exit
```

After the reboot, check the driver with the following command:

```
$ aplay -l
**** List of PLAYBACK Hardware Devices ****
card 0: ODROIDHDMI [ODROID-HDMI], device 0:
I2S.27 dit-hifi-0 []
  Subdevices: 1/1
  Subdevice #0: subdevice #0
card 1: ODROIDDAC [ODROID-DAC], device 0:
I2S.27 pcm5102-0 []
  Subdevices: 1/1
  Subdevice #0: subdevice #0
```

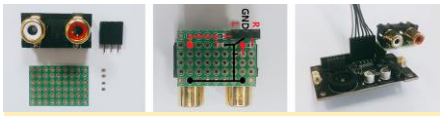


Figure 7 – Adding an RCA connector for line out to the Stereo Boom Bonnet

Schematics are available at https://goo.gl/pxTTd9, and detailed information may be found on the Wiki page at https://goo.gl/JY1Y7B.

Boom Box: Sound Engineering a Better Speaker

December 1, 2017 By @Technicavolous Tinkering, Tutorial

To improve the sound of Hardkernel's Stereo Boom Bonnet (**https://goo.gl/TrDU8u**), I went to my local hobby shop and got an "assortment pack" of styrene which happened to have some .080mil sheets and some tubing in it. I also purchased a little styrene solvent for welding. I did everything by eye, so some of the corners are not square, but this is intended to go into another cabinet, with only the face plate showing through. When it's complete, I'll glue a piece of cloth over the front and remount the speaker with hex head screws exposed. The boxes will be for acoustics and vibration control. If I were to do this as a standalone cabinet, I would probably make it about half as deep, but I chose this size because of the depth of the housing cabinet. I don't think there would be much sound difference with a longer tube.

One thing not shown in the pictures is shredded cotton balls. When the project is complete, I'll have the wires extend to a connector on the back, and the entire box will be loosely stuffed with pulled cotton to make a baffle, in order to prevent a "hollow box" sound.

I am deliberately leaving out measurements, since they don't matter much. I think it would be better to replace the tube with a simple flat sheet spaced up from the bottom about a quarter diameter of the speaker and spaced from the back about the same length. You can find designs all over the Internet for various chamber designs and ideas if you want to go more in-depth.

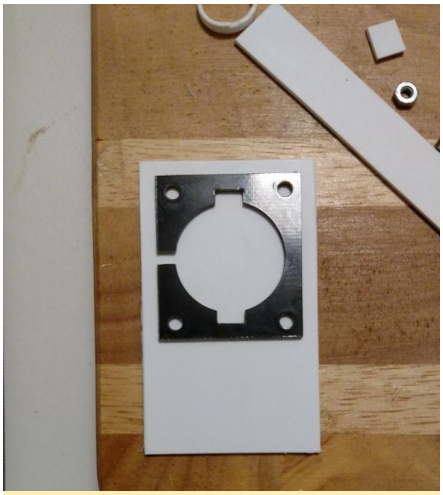


Figure 1 – I just set the little board over my cut panel and marked it with a sharpie

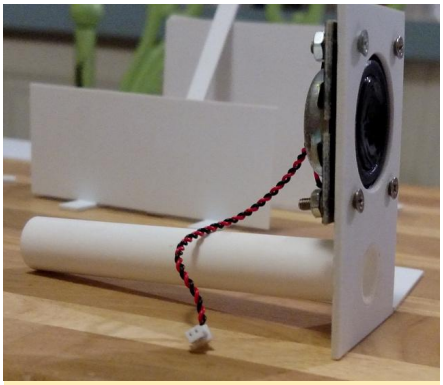


Figure 2 – Closeup of the tube attached

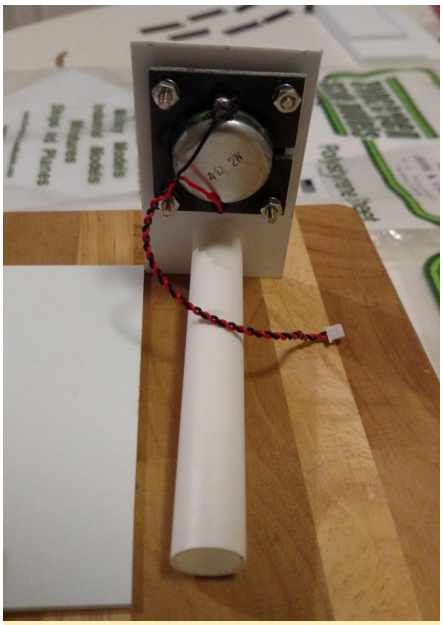


Figure 3 – Another view

Figure 4 – Top view with the front panel in the almost complete box

Figure 5 – The project ends up a sweet looking and great sounding box

Do something like this with any speaker, including the Stereo Boom Bonnet, and it will instantly sound better. To experiment with the sound, I inserted the front panel in backwards since I haven't extended the wires yet, and the sound was amazing. It tries to do a little bass, not much below 100Hz,but if I put my finger over the tube most of the bass goes away, so the chamber appears to be working like it should. Once the wires are extended out the back and the tube is in the box properly, it should give a tiny bit more bass.

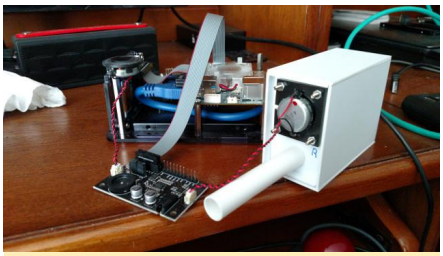
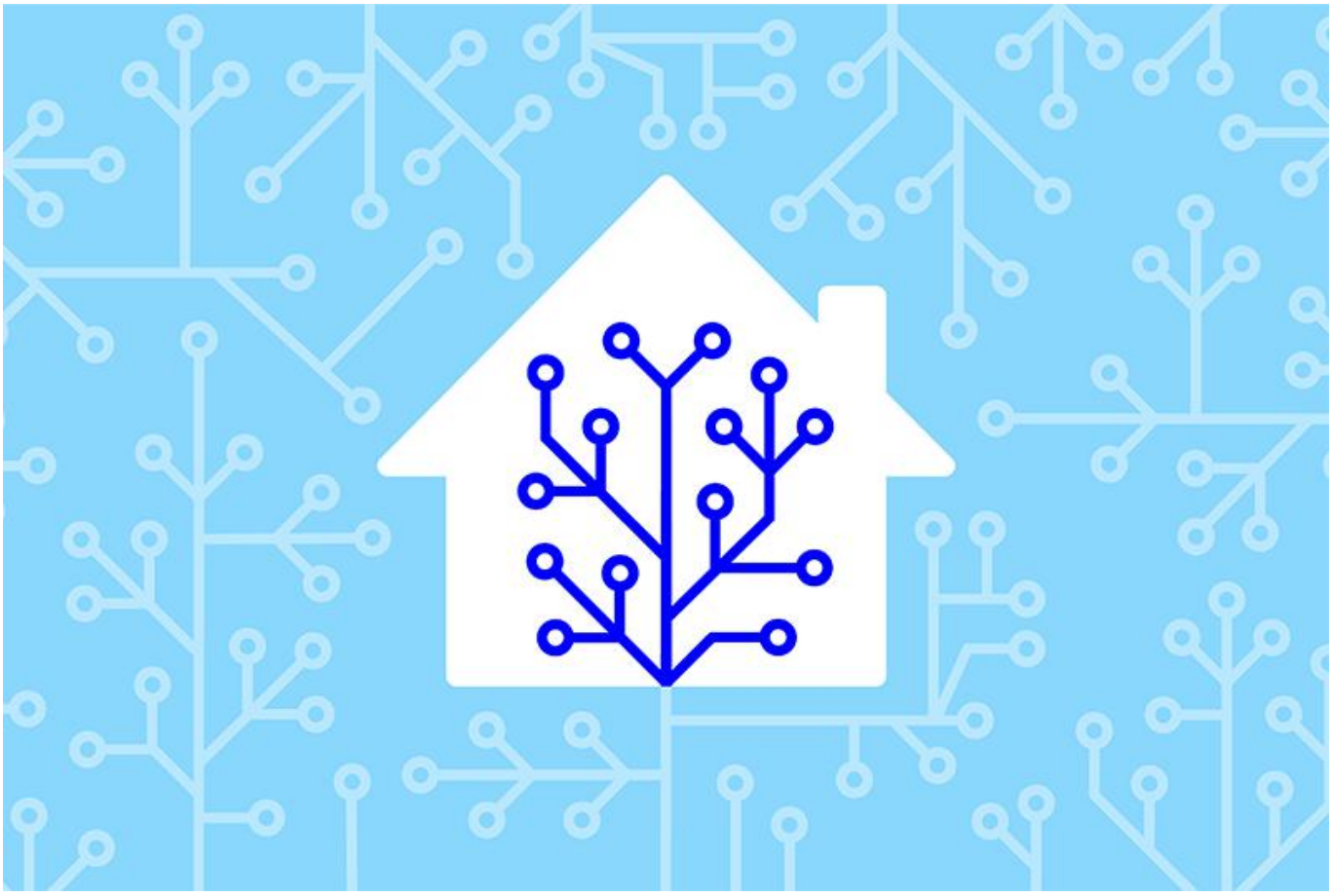


Figure 6 – Inserting the front panel backwards improved the bass response

For comments, questions and suggestions, please visit the original post at **https://goo.gl/yyp22n**.

Home Assistant: Using Infrared, Motors, and Relays

December 1, 2017 By Adrian Popa Tinkering



In this article, we are going to combine ODROIDs with a little bit of hardware and Home Assistant, so that we can start converting non-smart appliances to "smart" appliances. We'll be working with wires, doing a bit of soldering, connecting relays, and in some cases working with mains voltage, so be careful!

Converting an air conditioning unit into an IoT AC

If you have an older air conditioning (AC) unit, or if you're planning on buying a new model, you might want to be able to control it from anywhere. For example, you could turn on the AC from your phone when you're leaving work or when returning from a long vacation. You could get a WiFi-enabled AC, but those are about $200 more expensive than a non-WiFi unit. Instead, we're going to turn a non-WiFi AC, LG P12RL.NSB, into a smart one. This will be done by controlling the AC unit through an ODROID-XU4, an infrared (IR) blaster, and Home Assistant. The IR blaster can be used to control any device that has a remote control, not just an AC unit.

Figure 1 – Original AC remote (model AKB73456113)

The first thing you need to consider is how will the remote control communicate with the AC unit? There are two methods: the first is when pressing a button on the remote the whole state is sent via IR (temperature, fan speed, power state, etc). The other option is pressing a button only the current action is sent (increase temperature, turn on fan, etc). You can check this, for instance, by sending a command to turn on the fan to max speed, then turning off the fan, but with the remote out of range of the device, followed by issuing a different temperature change command. If the fan remains blowing at full speed, then the remote is not sending the full state. In case it is sending the full state, the project at http://bit.ly/2AcpKAW might help you decode the state information being sent. In my case, the IR remote sends only the current key being pressed, with the exception of the power-on message, which sends also the temperature and fan state.

The hardware

Designing and building an IR blaster for an ODROID board is relatively simple and is already documented on the wiki at http://bit.ly/2A6HHmR. I wanted my implementation to work on either an ODROID-XU4 or an ODROID-C2, so I needed to be able to power it from the 5V line and drive a transistor from the GPIO. The final assembly and circuit is shown in Figure 2.

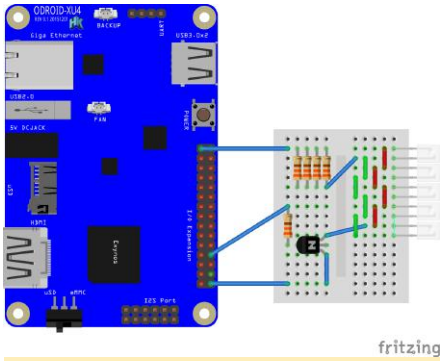


Figure 2 – Breadboard view of IR blaster

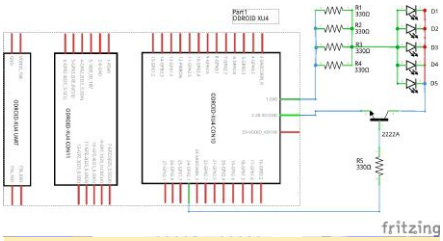


Figure 3 – Circuit diagram of IR blaster

I used an existing IR extender from my Samsung TV for the IR leds. After opening it up, I found out it had 5 LEDs wired in parallel, but I had no specifications for the LEDs. I estimated that each diode could pass a peak current of around 30 – 50mA (based on a common datasheet at http://bit.ly/1trG4ZM), so 5 of them could pass between 150 – 250mA. This is why I would need a resistor of about 82Ω to limit the current from the ODROID. If you're using

fewer IR LEDs, you will need a larger resistor. Sadly, such low resistors are hard to find, so I wired 4 330Ω resistors, from the C tinkering kit, in parallel instead (R1/R4). The transistor is controlled by pin 24 on the XU4, and will be used to modulate the signal. When the GPIO is turned on, the current flows and the IR LEDs turn on, and when the GPIO is off, the transistor shuts down the circuit. Resistor R5 is there to protect the GPIO.

Once you manage to build this, you can test the hardware by manually toggling GPIO 24 via sysfs and using a phone camera to film the leds. IR light should be visible on the camera and has a blue-ish hue. You can build it without a breadboard and you can even fit the transistor and resistors inside the XU4 case, but it will be a bit of a struggle. I broke the solder points twice while trying to fit it inside the case. The following piece of code can manually toggle GPIO pin 24:

```
$ sudo su -
# cd /sys/class/gpio/
# echo 24 > export
# cd gpio24
# echo out > direction
# echo 1 > value
# echo 0 > value
```

LIRC integration

Now, we need to tell LIRC that it can use the IR blaster to send data. To do this, use the following instructions in the wiki: **http://bit.ly/2A6HHmR**

```
$ sudo apt-get install lirc
$ sudo vi /etc/lirc/hardware.conf
#Chosen IR Transmitter
TRANSMITTER="ODROID blaster"
TRANSMITTER_MODULES="lirc_odroid lirc_dev"
TRANSMITTER_DRIVER=""
TRANSMITTER_DEVICE="/dev/lirc0"
TRANSMITTER_SOCKET=""
TRANSMITTER_LIRCD_CONF=""
TRANSMITTER_LIRCD_ARGS=""
```

Before restarting LIRC, we need to pass the correct parameters to the lirc_ODROID module. We can do this by creating the file /etc/modprobe.d/lirc.conf with the following contents:

```
options lirc_odroid gpio_out_pin=24
softcarrier=1 invert=0
```

Try restarting LIRC and monitor dmesg for any errors:

```
$ sudo systemctl enable lirc
$ sudo service lirc restart
```

In case dmesg shows errors like below, it means that some other driver has claimed GPIO PIN 24.

```
[   25.322482] lirc_dev: IR Remote Control
driver registered, major 245
[   25.336230] lirc_odroid: module is from
the staging directory, the quality is
unknown, you have been warned.
[   25.346335] lirc_odroid: cant claim gpio
pin 24
[   25.350461] lirc_odroid: init port fail!
[   25.353337] lirc_odroid[lirc_ODROID_exit]
```

In my case, it was the 1wire module, which can be disabled by adding the following lines to /etc/modprobe.d/blacklist-odroid.conf and rebooting:

```
# 1 wire
blacklist w1_gpio
blacklist wire
```

Getting the remote control codes

Once the transmitter seems to be ready, it's time to get the remote control codes for your remote. The codes consist of a series of intervals when the signal is on or off measured in microseconds. You can get these codes from various online sources, or you can record them with LIRC and an IR receiver. I used the IR receiver on a C2 to record each code to a different file, pressed CTRL+C to exit mode2, and cleaned it up by deleting the first row. When I was done, I added entries to lircd.conf:

```
c2$ sudo apt-get install lirc
c2$ sudo service lirc stop
c2$ sudo mode2 -m -d /dev/lirc0 | tee power-
on
c2$ sudo sed -i '1d' power-on
```

This can get tedious, because I had to record the codes for power-on/off, temperature-18 through temperature-30, fan-low, fan-med, fan-high, swing-on/off, jet-on/off, ionizer-on/off. However, once I did, I merged them to a config file (/etc/lirc/lircd.conf) as demonstrated at **http://bit.ly/2A9MK3r**. You can restart LIRC and test that you are able to send the codes with irsend:

```
$ sudo service lirc restart
$ irsend LIST lgirplus.conf ""
$ irsend SEND_ONCE lgirplus.conf power-on
$ irsend SEND_ONCE lgirplus.conf power-off
```

In case the last irsend command fails/times-out, you may have run into a LIRC/driver bug. The quick fix is to restart LIRC before injecting each command.

Integration in Home Assistant

If the LIRC blaster is connected to the same device where you have Home Assistant installed, you could use the Shell Command component (**http://bit.ly/2vOFnhe**) to issue IR commands from HA. In my case, LIRC was running on a different system than HA, so I developed a Python script that talks to HA through MQTT and issues the irsend commands.

The complete code for this MQTT agent is available at **http://bit.ly/2Ax8SYz**. The code gets configuration data from /etc/ir-ac-mqtt-agent.yaml, then connects with MQTT to the broker. For a guide on setting up the broker and Home Assistant, refer to the ODROID magazine article at **http://bit.ly/2A6ql9I**. It also defines two callback functions:

- on_connect – registers to a list of MQTT topics to listen for commands
- on_message – gets called each time a MQTT message is received for the registered topics.

The script also keeps an internal dictionary with the current state, as such power-on and temperature. This allows it to better react to incoming commands. For instance, if it receives a power off command, but the power is already off, it will ignore the command to avoid the AC unit from beeping.

The logic inside tries to simulate what the physical remote does by ignoring commands unless the power is on, and also setting the temperature to 18C and fan to full when enabling "Jet Mode". There were some simplifications done as well, such as when sending the power-on command, the temperature is set to 21C and the fan to high because the power-on signal encodes some of the state of the remote. The agent script listens to commands issued on topics such as ha/lg_ac/ionizer/set and sends feedback on ha/lg_ac/ionizer/get, so that the web interface has feedback that the command was received.

To install the MQTT agent, you can use these commands:

```
$ sudo wget -O /usr/local/bin/ir-ac-mqtt-
agent.py
https://raw.githubusercontent.com/mad-
ady/home-assistant-
customizations/master/external-scripts/ir-
ac-mqtt-agent.py
$ sudo chmod a+x /usr/local/bin/ir-ac-mqtt-
agent.py
$ sudo apt-get install python-pip python-
yaml
$ sudo pip install paho-mqtt
$ sudo wget -O /etc/ir-ac-mqtt-agent.yaml
https://github.com/mad-ady/home-assistant-
customizations/blob/master/external-
scripts/ir-ac-mqtt-agent.yaml
$ sudo wget -O /etc/systemd/system/ir-ac-
mqtt-agent.service
https://github.com/mad-ady/home-assistant-
customizations/blob/master/external-
scripts/ir-ac-mqtt-agent.service
```

Take your time to make the necessary changes to /etc/ir-ac-mqtt-agent.yaml, then enable and start the service:

```
$ sudo systemctl enable ir-ac-mqtt-agent
$ sudo systemctl start ir-ac-mqtt-agent
```

On the Home Assistant side, we will configure several MQTT switches (**http://bit.ly/2AwtDUd**) to handle Power, Jet mode, Ionizer, Swing, an input_select (**http://bit.ly/2zEfgNA**), to select fan speed mode and an input_number (**http://bit.ly/2k0vfOY**), to hold the desired temperature. The switches communicate their state with the backend script via MQTT directly, while the other components make use of automation to trigger MQTT messages on changes. Here is the component configuration:

```
switch:
  - platform: mqtt
    command_topic: 'ha/lg_ac/power/set'
    state_topic: 'ha/lg_ac/power/get'
    payload_on: 'ON'
    payload_off: 'OFF'
    name: 'AC Power'
    retain: false
  - platform: mqtt
    command_topic: 'ha/lg_ac/ionizer/set'
    state_topic: 'ha/lg_ac/ionizer/get'
    payload_on: 'ON'
    payload_off: 'OFF'
    name: 'AC Ionizer'
    retain: false
  - platform: mqtt
    command_topic: 'ha/lg_ac/jet/set'
    state_topic: 'ha/lg_ac/jet/get'
    payload_on: 'ON'
    payload_off: 'OFF'
    name: 'AC Jet'
    retain: false
  - platform: mqtt
    command_topic: 'ha/lg_ac/swing/set'
    state_topic: 'ha/lg_ac/swing/get'
    payload_on: 'ON'
    payload_off: 'OFF'
    name: 'AC Swing'
    retain: false

input_select:
  lg_ac_fan_mode:
    name: Fan mode
    options:
      - cycle
      - low
      - med
      - high
    initial: 'low'

input_number:
  lg_ac_temperature:
    name: AC Temperature
    initial: 22
    min: 18
    max: 30
    step: 1
```

We can group all of these elements in a separate view:

```
group:
…
 lg_ac:
    name: Air Conditioning
    view: yes
    icon: mdi:snowflake
    entities:
      - group.lg_ac_group
 lg_ac_group:
    name: LG AC
    entities:
      - switch.ac_power
      - input_number.lg_ac_temperature
      - input_select.lg_ac_fan_mode
      - switch.ac_jet
      - switch.ac_ionizer
      - switch.ac_swing
```

And after restarting Home Assistant, it should look like Figure 4.

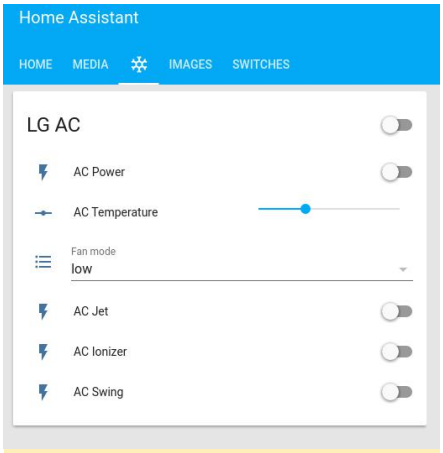


Figure 4 – Basic support for air conditioning

```
- action:
  - alias: LG AC MQTT Set Temperature
    data:
      payload_template: '{{
states.input_number.lg_ac_temperature.state
}}'
      qos: 0
      retain: true
      topic: ha/lg_ac/temperature/set
    service: mqtt.publish
  alias: LG AC Set IR temperature
  id: '1499081218012'
  trigger:
  - entity_id:
input_number.lg_ac_temperature
    platform: state
- action:
  - alias: LG AC MQTT Set Fan
    data:
      payload_template: '{{
states.input_select.lg_ac_fan_mode.state }}'
      qos: 0
      retain: true
      topic: ha/lg_ac/fan/set
    service: mqtt.publish
  alias: LG AC Set IR Fan
  id: '1499152161'
  trigger:
  - entity_id: input_select.lg_ac_fan_mode
    platform: state
- action:
  - alias: LG AC Set temperature slider
    service: input_number.set_value
    data_template:
      entity_id:
input_number.lg_ac_temperature
      value: '{{trigger.payload}}'
  alias: LG AC Read temperature via MQTT
  id: '1499423002'
  trigger:
  - platform: mqtt
    topic: ha/lg_ac/temperature/get
- action:
  - alias: LG AC Set fan combo box
    service: input_select.select_option
    data_template:
      entity_id: input_select.lg_ac_fan_mode
      option: '{{trigger.payload}}'
  alias: LG AC Read temperature via MQTT
  id: '1499423003'
  trigger:
  - platform: mqtt
    topic: ha/lg_ac/fan/get
```

The first two automations push the values for the temperature and fan components via MQTT on state change, while the last two automations receive temperature and fan data through MQTT to update the web interface. Restarting again should give you a functional AC system controlled by Home Assistant. Here's a video of an early prototype of it in action: https://youtu.be/zGRlhILVRCQ.

Adding a start & stop timer

The original AC remote has an option to start and stop the AC on a timer. We can also model that inside Home Assistant with a few automations and some extra components.

Ideally you would use the input_datetime component (http://bit.ly/2A8Mmoc), to allow the user to select the start and stop times, but at the time of writing this article, the component is not fully working. On HA 0.56, so we will be using input_text instead (http://bit.ly/2i8lcXI), with a regular expression that allows the typing of a time. There are also two input_booleans (http://bit.ly/2Bnd4Yj), that look like switches and allows the user to enable/disable the functionality. Here is the configuration that goes into configuration.yaml:

```
input_text:
  lg_ac_on_timer:
    name: LG AC on timer
    initial: '16:00'
    pattern: '^[0-9]{1,2}:[0-9]{1,2}$'
  lg_ac_off_timer:
    name: LG AC off timer
    initial: '18:00'
    pattern: '^[0-9]{1,2}:[0-9]{1,2}$'

input_boolean:
  ac_on_timer_active:
    name: Activate AC On timer
    initial: off
    icon: mdi:calendar
  ac_off_timer_active:
    name: Activate AC Off timer
    initial: off
    icon: mdi:calendar

group:
…
  lg_ac:
…
    entities:
…
      - group.lg_ac_timer
  lg_ac_timer:
    name: AC Timer
    entities:
      - input_boolean.ac_on_timer_active
      - input_text.lg_ac_on_timer
      - input_boolean.ac_off_timer_active
      - input_text.lg_ac_off_timer
```

The end result after restarting Home Assistant looks like Figure 5.

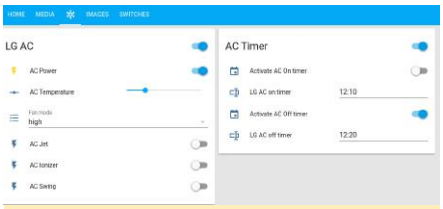


Figure 5 – AC controls with timers

To make it work, you will need to add the following two automations in automations.yaml. The first automation turns on AC, Jet mode and also turns off the Activate AC On timer, so that it is a one-shot event. It checks every minute to see if the ac_on_timer_active is on and if the current time is the same as the string inside lg_ac_on_timer. The second automation does a similar thing for the off timer, but with different actions.

```
- action:
  - service: switch.turn_on
    entity_id:
    - switch.ac_power
  - service: switch.turn_on
    entity_id:
    - switch.ac_jet
  - service: input_boolean.turn_off
    entity_id:
    - input_boolean.ac_on_timer_active
  alias: Turn On AC on timer
  id: '1502194970'
  trigger:
  - platform: time
    minutes: /1
    seconds: 0
  condition:
  - condition: state
    entity_id:
input_boolean.ac_on_timer_active
    state: 'on'
  - condition: template
    value_template: '{{
now().strftime("%H:%M") ==
states.input_text.lg_ac_on_timer.state
      }}'
- action:
  - service: switch.turn_off
    entity_id:
    - switch.ac_power
  - service: input_boolean.turn_off
    entity_id:
    - input_boolean.ac_off_timer_active
  alias: Turn Off AC on timer
  id: '1502194971'
  trigger:
  - platform: time
    minutes: /1
    seconds: 0
  condition:
  - condition: state
    entity_id:
input_boolean.ac_off_timer_active
    state: 'on'
  - condition: template
    value_template: '{{
now().strftime("%H:%M") ==
states.input_text.lg_ac_off_timer.state
      }}'
```

After restarting Home Assistant once again, you should have a complete "Smart" AC system which you can enjoy during heat waves. Although the automations above are pretty basic, you can add more logic – like adding a thermostat (http://bit.ly/2Ay6kJN), and monitor the outside weather either from a sensor, or from the weather forecast, so you can turn on or of your AC when the outside weather is over a threshold. In fact, we will play with such a thermostat for our next project.

Controlling a gas heater with a Home Assistant Thermostat

Since for many readers it's winter, you may be interested in having a smart heater. We are going to control a natural gas boiler connected to the central heating, a Viessman Vitopend 100 (http://bit.ly/2Bd9X4d). The heater, if turned on, will try to keep a constant water temperature for the water that flows through the heating elements in the house, but this can be wasteful if nobody is home or if the house is well insulated. Instead, you should use a thermostat to turn the heater on or off based on the ambient temperature. My gas heater had a thermostat, Salus 091FLRF (http://bit.ly/2AbcnkC). My thermostat does the job, but it has some big drawbacks. For starters, it would reset in the middle of the night and lose the configuration, leaving me in the cold. An equivalent Internet-enabled thermostat is at least $100, so a DIY approach pays off.

The advantage of the external thermostat is that it had a relay connected to the gas heater and I didn't have to open it up. If you do have to open up your heater, make sure you call a specialized technician, or you could get in trouble with your natural gas provider.

After analyzing the thermostat schematics, it became apparent that communication with the heater is done by closing or opening a relay on a 220V line. If we add a second relay in parallel with the first one I can close either of them to turn the heater on or off. Why do you need a relay? So that you don't fry your ODROID board! The plan is to use a Sainsmart 2 Channel relay (http://bit.ly/2A5WEFF), to connect the ODROID to the 220V line that goes into the heater. Even if the relay is rated for 5V, it can be safely used with an ODROID-C1 or C2's 3.3V GPIOs.

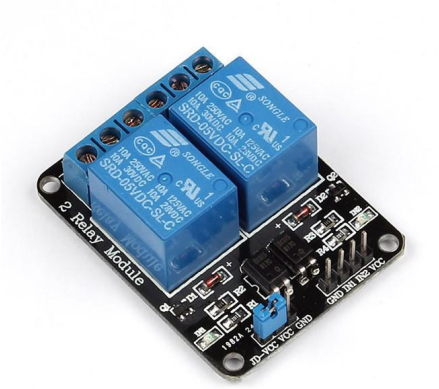


Figure 6 – SainSmart 2 Channel Relay

Since you will be working with mains voltage, make sure you either get a certified electrician or comply to the laws in your country. Also always disconnect all appliances from mains when working with these lines. The schematic we're going to implement is pretty simple as shown in Figure 7.

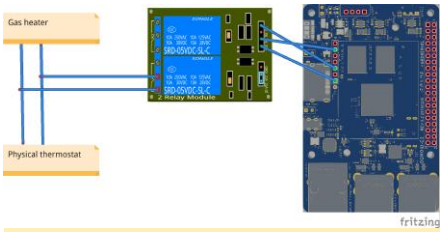


Figure 7 – Implementation details

There is only one complication with this setup. Instead of using the 40 pin GPIO connectors on the C2 (J2 header), I used some pins from the I2S header (J7) instead. The reason is, in my case the J2 header was used by HardKernel's 3.5" display shield (http://www.hardkernel.com/main/products/prdt_info.php?g_code=G147435282441).

In order to convert the I2S pins into GPIO pins, you will need to edit the DTB loaded with the kernel. Follow the instructions below to edit the DTB, and also make the changes persistent when you upgrade the kernel through apt, as discussed at http://bit.ly/2AyrWWz.

```
$ sudo apt-get install device-tree-compiler
$ sudo fdtput
/media/boot/meson64_ODROIDc2.dtb /I2S status
disabled
$ sudo fdtput
/media/boot/meson64_ODROIDc2.dtb
/i2s_platform status disabled
$ sudo vi /etc/kernel/postinst.d/i2s-disable
#!/bin/bash

echo "Disabling I2S support from the device
tree"
/usr/bin/fdtput
/media/boot/meson64_ODROIDc2.dtb /I2S status
disabled
/usr/bin/fdtput
/media/boot/meson64_ODROIDc2.dtb
/i2s_platform status disabled
```

Once you reboot, the J7 header will have only GPIO pins that you can use.

Controlling the relay through MQTT

Since it is not the ODROID that runs Home Assistant, we will need to be able to control it over the network, with MQTT. The agent code listens for ON/OFF messages on a specific topic and turns the GPIO on or off. Normally, we would have used wiringPI (http://bit.ly/2zBMm0F), but in this case the J7 connector is not mapped to wiringPI, so we will be using sysfs instead. The methods pinMode and digitalWrite emulate their wiringPI counterparts to make the code easier to understand. The code is available at http://bit.ly/2jofKfN. For brevity the code is not included here. To install it on your system, follow these steps:

```
$ sudo wget -O /usr/local/bin/heater-mqtt-
agent.py
https://raw.githubusercontent.com/mad-
ady/home-assistant-
customizations/master/external-
scripts/heater-mqtt-agent.py
$ sudo chown a+x /usr/local/bin/heater-mqtt-
agent.py
$ sudo wget -O /etc/heater-mqtt-agent.yaml
https://github.com/mad-ady/home-assistant-
customizations/blob/master/external-
scripts/heater-mqtt-agent.yaml
$ sudo wget -O /etc/systemd/system/heater-
mqtt-agent.service
https://github.com/mad-ady/home-assistant-
customizations/blob/master/external-
scripts/heater-mqtt-agent.service
$ sudo apt-get install python-pip python-
yaml
$ sudo pip install paho-mqtt
```

Edit the configuration file at /etc/heater-mqtt-agent.yaml, set your MQTT details, then start the agent with:

```
$ sudo systemctl enable heater-mqtt-agent
$ sudo systemctl start heater-mqtt-agent
```

You can monitor messages from the agent with the following command:

```
$ sudo journalctl -f -u heater-mqtt-agent
```

Home Assistant integration

To make use of this agent inside Home Assistant, you can configure a MQTT Switch and add it inside its own group:

```
switch:
…
 - platform: mqtt
    command_topic: 'ha/heater/set'
    state_topic: 'ha/heater/get'
    payload_on: 'ON'
    payload_off: 'OFF'
    name: 'Heater'
    retain: true

group:
...
  heater:
    name: Gas heater
    view: yes
    icon: mdi:fire
    entities:
      - switch.heater
```

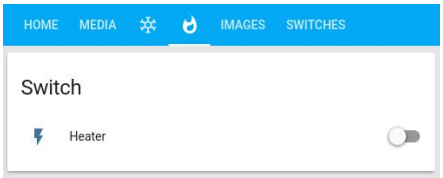


Figure 8 – A switch for your heater

Having a switch that you can toggle is nice, but you probably want a thermostat. A thermostat tries to keep a temperature between certain levels, and Home Assistant requires a temperature sensor and a switch (http://bit.ly/2Ay6kJN). The temperature sensor could be any sensor, including the outside temperature, but in our case we may want to use data from sensors in multiple rooms. I have two DS18b20 temperature sensors in two rooms already integrated into Home Assistant, as described in my previous ODROID article at http://bit.ly/2A6ql9I. However, the thermostat can act only on one temperature sensor. We will need to combine the two sensors into one sensor, which returns the minimum temperature between the two. This can be easily extended to multiple sensors, so that the thermostat heats the coldest room to the desired temperature. We can do this with a template sensor (http://bit.ly/2wPQLeY) inside configuration.yaml:

```
sensor:
…
  - platform: template
    sensors:
…
     house_temperature:
        friendly_name: Minimum house
temperature
        unit_of_measurement: '_C'
        value_template: '{{
(states.sensor.temperature_rest_python.state
, states.sensor.temperature_via_mqtt.state)
| min }}'

group:
…
  weather:
…
    entities:
…
      - sensor.house_temperature
```

Adding the thermostat is now an easy task (configuration.yaml):

```
climate:
  - platform: generic_thermostat
    name: Heater thermostat
    heater: switch.heater
    target_sensor: sensor.house_temperature
    min_temp: 15
    max_temp: 30
    target_temp: 24
    min_cycle_duration:
      minutes: 5
```

```
    tolerance: 0.3

group:
…
  heater:
…
    entities:
…
      - climate.heater_thermostat
```

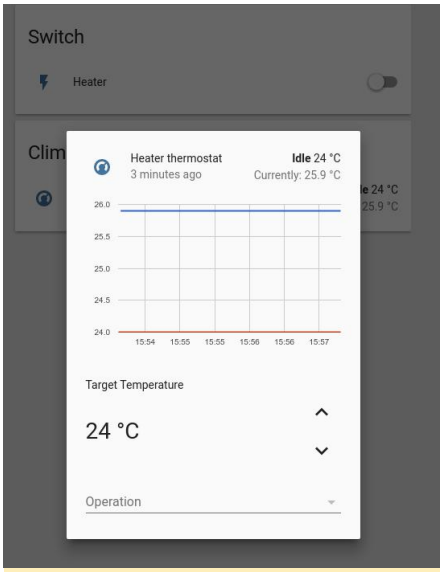


Figure 9 – Thermostat

The thermostat configuration will turn on the heat when the temperature is below target_temp – tolerance, will keep the heater on for at least min_cycle_duration, then will turn it off when the temperature is above target_temp + tolerance. You can use a similar thermostat with the AC we've built before, but you need to configure it with ac_mode: true so that it is a cooling device.

You can manually control/set the thermostat, but the fun part is that you can use automations to control the thermostat. A simple use case is to set different target temperatures based on presence detection or time of day. The following automations set the target temperature to 23C during the day and 25.5C during the night (automations.yaml):

```
- action:
  - alias: climate.set_temperature
    data:
      entity_id: climate.heater_thermostat
      temperature: 23
    service: climate.set_temperature
  alias: Thermostat set low
  condition: []
  id: '1506943539788'
  trigger:
  - at: 07:00
    platform: time
- action:
  - alias: climate.set_temperature
    data:
      entity_id: climate.heater_thermostat
      temperature: 25.5
    service: climate.set_temperature
  alias: Thermostat set high
  condition: []
  id: '1506943638481'
  trigger:
  - at: '19:00'
    platform: time
```

Controlling a window blind with Home Assistant

One more physical item we can integrate into Home Assistant is a motorized window blind. In my case, the motors were controlled by a three-way physical switch. When in the up position, the blind raises. In the middle position, the motors are off, and in the down position it lowers. I wanted to use a relay and take over the slide operation when the physical switch is in the middle position, but also to allow manual operation with the regular switch. After a long discussion on the forum (http://bit.ly/2AwFCBb), I settled on the physical implementation shown in Figure 10.

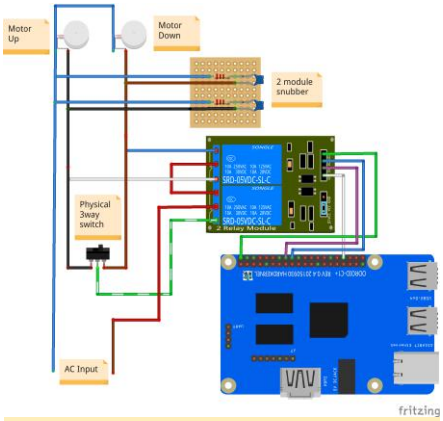


Figure 10 – Controlling a window blind motor

Apart from the ODROID-C1+ and the 2-module relay board, we also need to use a couple of snubbers. The snubbers are RC circuits which have to discharge the reactive energy that builds up in the motors during switching, otherwise transient power spikes will brown-out your relay contacts when they try to discharge, I damaged a relay before realizing this. I used the types of snubbers described at http://bit.ly/2A58tf4 in my build.

Please note that while my design minimizes the number of components used, it's more risky because it operates at 220V. Forum user @Jojo proposed an alternate design which changes the physical switch's voltage to 5V and uses digital circuits to drive the relay, which should be safer (http://bit.ly/2Bf2txo).

MQTT agent for blind operation

As you've grown accustomed already, we need a script running on the ODROID that listens to commands for the blinds, sent through MQTT, then executes the commands by triggering the relay and reports back the status. The difference from previous code shown is that this time we will have to use threads, more specifically timers. The simple way the code can operate is:

A message to OPEN or CLOSE the blind is sent from Home Assistant
The script turns on the first relay and assumes control of the motors, then uses the second relay to control the direction the blind operates (up or down)
The script waits for a set number of seconds so that the blind can fully open or close, which takes 17 seconds in my case
When the time expires, the relay is reset to manual mode and a status update is sent via MQTT back to Home Assistant

The complication comes if you want to stop the blind midway, or at an arbitrary location. To do this, Home Assistant can send a STOP command, or a desired position, but if the script is single-threaded, it is stuck waiting for the motor to finish before processing the STOP command.
By using timers, after the motor is started, a timer is scheduled to stop the motor in 17 seconds, and the code goes back to listening for MQTT messages. If a STOP message comes before the timer expires, it will cancel the timer and run the stopBlinds() command immediately. Additionally, the script keeps an internal state of the blind position and the direction of the motor, so that if the blind is midway, and you want to open it 70%, it will be able to calculate for how long it needs to run the motor and in which direction. The code is available at http://bit.ly/2A88Ohk and can be installed with:

```
$ sudo wget -O /usr/local/bin/blind-cover-
mqtt-agent.py
https://raw.githubusercontent.com/mad-
ady/home-assistant-
customizations/master/external-
scripts/blind-cover-mqtt-agent.py
$ sudo chown a+x /usr/local/bin/blind-cover-
mqtt-agent.py
$ sudo wget -O /etc/systemd/system/blind-
cover-mqtt-agent.service
https://raw.githubusercontent.com/mad-
ady/home-assistant-
customizations/master/external-
scripts/blind-cover-mqtt-agent.service
$ sudo wget -O /etc/blind-cover-mqtt-
agent.yaml
https://raw.githubusercontent.com/mad-
ady/home-assistant-
customizations/master/external-
scripts/blind-cover-mqtt-agent.yaml
$ sudo apt-get install python-pip python-
yaml
$ sudo pip install paho-mqtt
```

In addition to this, you will need to install wiringPi library and its Python bindings (http://bit.ly/2AwVQui). You will need to edit /etc/blind-cover-mqtt-agent.yaml and input your MQTT details, then you can enable and start the agent:

```
$ sudo systemctl enable blind-cover-mqtt-
agent
$ sudo systemctl start blind-cover-mqtt-
agent
```

You will be able to view debug messages with the following command:

```
$ sudo journalctl -f -u blind-cover-mqtt-
agent
```

Home Assistant configuration

Home Assistant comes with a MQTT cover component (http://bit.ly/2jmAlf7), which provides a basic interface for the cover. You can configure it with this configuration inside configuration.yaml:

```
cover:
  - platform: mqtt
    name: "Blinds"
    state_topic: "ha/blind_cover/get"
    command_topic: "ha/blind_cover/set"
    set_position_topic:
"ha/blind_cover/position"
    assumed_state: true
    payload_open: 'OPEN'
    payload_close: 'CLOSE'
    payload_stop: 'STOP'
    state_open: 'open'
    state_close: 'closed'
group:
…
 blinds:
    name: Blinds
    view: yes
    icon: mdi:blinds
    entities:
      - cover.blinds
```

After restarting Home Assistant, the blind control will look like Figure 11:

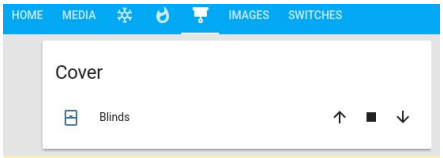


Figure 11 – Standard blind controls

At https://youtu.be/MlhuELv1244, you can see a demo of the blinds. As I mentioned before, is there a way to set an arbitrary position for the blind? There is a way to tweak the blind component to display a slider next to the blind component that allows you to control its position. To do this, you will need to install the Home Assistant custom UI, downloaded from **http://bit.ly/2AcqDJE**, and type the following commands:

```
$ sudo su - homeassistant
$ cd .homeassistant/
$ curl -o update-custom-ui.sh
"https://raw.githubusercontent.com/andrey-
git/home-assistant-custom-
ui/master/update.sh?raw=true"
$ chmod a+x update-custom-ui.sh
$ ./update-custom-ui.sh
```

The configuration needs to be tweaked a bit to load the customized blind control. First, you will need to activate the custom UI controls by making these changes to configuration.yaml:

```
customizer:
  custom_ui: local

Use the customize section to specify that
all covers should use the new UI (inside
configuration.yaml):

homeassistant:
  customize_glob:
    cover.*:
      custom_ui_state_card: state-card-
custom-ui
```

By default, the cover will look the same as the old one, so we need to manually enable the slider (under the customize section in configuration.yaml):

```
homeassistant:
  customize:
    cover.blinds:
      state_card_mode: break-slider
      stretch_slider: true
```

After restarting Home Assistant, the user interface will look like Figure 12.

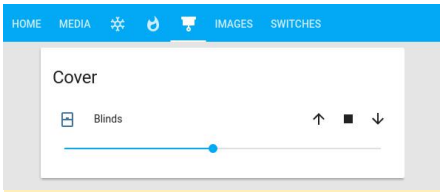


Figure 12 – Cover with slider

As you can see, with a bit of work you can turn your ordinary house into a smart house with the help of ODROIDs and Home Assistant. For feedback and discussions, please visit the original thread at **http://bit.ly/2s13GbB**.

Android Gaming: Stranger Things, Pocket Morty, and Streets of Rage

December 1, 2017 By Bruno Doiche Android, Gaming

It's easy to chill with your ODROID because there are so many Android gaming options for the Android operating system. In this article, we'll explore three new releases: Stranger Things: The Game, Pocket Morty's, and Streets of Rage.

Stranger Things: The Game

Straight from the hit TV show, here comes a game with an unique feel. The feeling that when there is someone backing up a game studio, you will have the best fun without in-app purchases, advertisements or those pesky banners!

Stranger Things: The Game has a color palette that is reminiscent of a classic SNES box

But let us talk about the game itself, which is a bucket of references from the 1980s, although the game itself looks like it has been pulled from the classic 1990s adventures that we played over and over on our VGA PCs or Amigas of old. You control a bunch of characters from the series going around the town of Hawkins to solve the big mystery hidden in it, each with unique skills that you will be certainly using to solve puzzles and defeat enemies. The controls are the simplest possible, so there are no worries about having them working on your ODROID touchscreen or your USB controller. You can map the controls to work with the keyboard, but I didn't test that feature.

The game follows a freestyle approach from the series

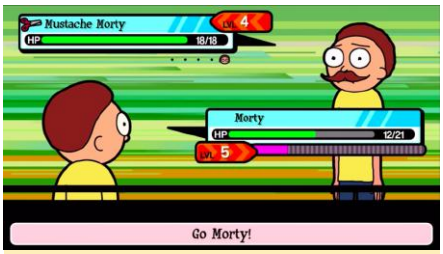
With various chapters, if you are the average player, you will take from five to ten hours to complete if you don't get stuck on a weird side quest. But if this is too easy for you, you can go for the classic mode, which describes itself as 1980s hard, which is a little bit of a stretch since back in the 1980s, you would use a passcode to go throughout the chapters. Usually 1980s games only offered 3 hit points, 2 lives and, if the game was benevolent, a couple of continues! Nevertheless, the game is super catchy and fun. You will have a blissful time collecting items and essentially having a pretty decent game based on a TV show, which is a rarity nowadays.

The later chapters are no picnic, but if I can make it through, you can too

Above all, Stranger Things: The Game makes me fond of an era when the way that you would keep enjoying your favorite shows was to have an NES cartridge and play it back and forth with your friends during your vacations, which was the best!

Pocket Mortys

Straight out of the gate, remember when Pokémon Go was all the rage, and we had to teach you guys how to to spoof the location of your ODROID in order to be able to play that game more easily? What if you could play a similar game, without having to disappoint yourself by losing EVERY SINGLE TIME trying to do Gym battles and walking through your city having your hopes dashed by that pesky thousandth zubat? Pocket Mortys is the game that captures the original Pokémon spirit in a package that expands on the greatest, shiniest, and wildest cartoon that resonates with viewers like a neutron bomb this year.



Have you ever seen such a glorious moustache? No? Then fight him Morty!

The super original plot? You will play a Pokémon game, but with Rick and Morty characters. You capture and collect different versions of Morty, the co-protagonist of Rick and Morty, from the many alternate dimensions, leveling up them to get your portal gun back and for the fun of collecting Mortys. It is so close to Pokémon that I really advise you to get this game before Nintendo's lawyers figure out that it is a 1 to 1 reproduction. And you of course can enjoy it, for as simple as it may appear, it is in fact quite complicated to have a good Pokémonesque game. It is mischievously smart for a free-to-play game.

The game is so well designed that you will swear that you have seen the characters sporting those outfits in the show

Absurdly rewarding despite the rock-paper-scissors gameplay, this is a game that capture the feel from the TV show so well that we can almost say that with this game you can pretty much not get to watch the show! (Which is a lie; you should watch this show over and over and over)

The funnest part of this game, if you snap out of it, is seeing you are handling a gadget within a gadget, which is super meta

Streets of Rage: Android Edition

I really shouldn't have to present this game for anyone that is reading this I believe, but, come on, this is Streets of Rage. If you lived through the magical era of the early 1990s, when 16-bit gaming roamed the earth, you surely played this game. Of course, you can emulate this on your ODROID using RetroArch, but what if you are unable to find this ROM for your emulator? Well, now you are in luck!

Playing with multiple characters was the best thing in the 1990s

In the vein of what was the brawling beat-em-up games of the era, you have the archetype of the corrupt city that needed cops to do justice with their bare hands in order to defeat the city mafia and restore the safety of the population.

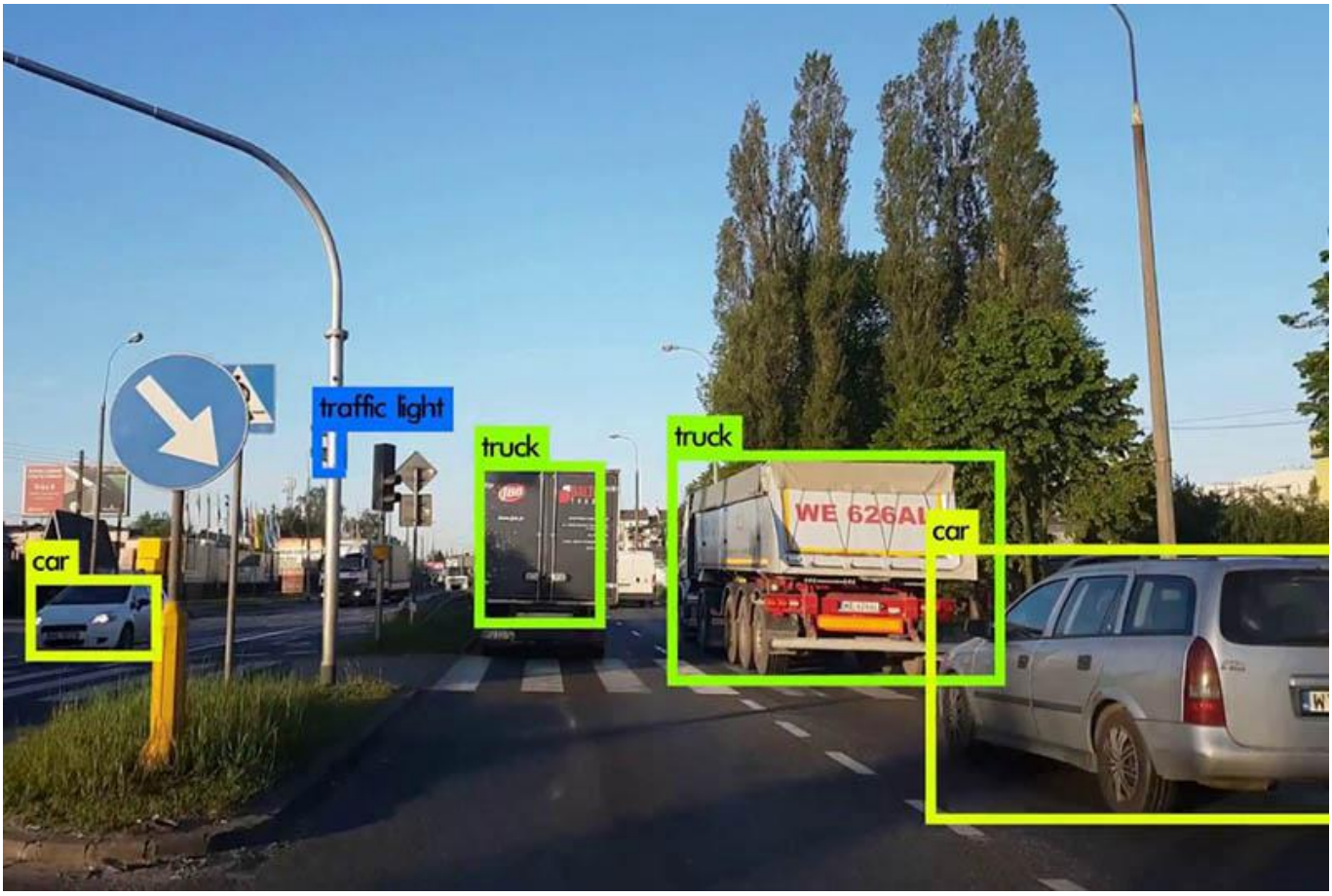
This start screen song and splash screen is etched on every 1990s gamer mind

With what is considered a masterpiece of fun game play and with Yuzo Koshiro's unforgettable soundtrack, this game is by itself the blueprint of what every retro gaming indie developers try to recreate when doing games.

Nothing was more satisfying than hitting the boss with the backup bazooka

Running YOLO On ODROID: YOLODROID

December 1, 2017 By Tom Jacobs Tinkering



YOLO (https://pjreddie.com/darknet/yolo/) is a neural network model that is able to recognise everyday objects very quickly from images. There's also TinyYOLO (http://machinethink.net/blog/object-detection-with-yolo/), which runs well on mobile devices. This guide tells you how to get TinyYOLO installed and running on your ODROID-XU4. To follow along, login to your ODROID, and run the commands show in the sections below.

Install TensorFlow

First, we make sure everything is up to date:

```
$ sudo apt-get update
$ sudo apt-get upgrade -y
$ sudo apt-get dist-upgrade -y
$ sudo reboot
```

Get some swap

Bazel won't build without using swap memory on the ODROID-XU4. Pop in a blank 8GB USB drive, which will get erased, and run the following command:

```
$ sudo blkid
```

Check the device name, usually /dev/sda1, and with that name, run:

```
$ sudo mkswap /dev/sda1
$ sudo swapon /dev/sda1
$ sudo swapon
```

Install the requirements

We'll need real Oracle Java, instead of OpenJDK. I tried OpenJDK, built Bazel with it, but it failed to SHA-1 hash downloads, and so was useless. So, we need to install the following packages:

```
$ sudo apt-get install pkg-config zip g++
zlib1g-dev unzip
$ sudo apt-get install gcc-4.8 g++-4.8
$ sudo update-alternatives --install
/usr/bin/gcc gcc /usr/bin/gcc-4.8 100
$ sudo update-alternatives --install
/usr/bin/g++ g++ /usr/bin/g++-4.8 100
$ sudo apt-get install python-pip python-
numpy swig python-dev
$ sudo pip install wheel
$ sudo add-apt-repository
ppa:webupd8team/java
$ sudo apt-get update
$ sudo apt-get install oracle-java8-
installer
$ sudo apt-get install oracle-java8-set-
default
$ java -version
```

Install Bazel build system

Google builds things using Bazel. TensorFlow is from Google. Thus, we need to build Bazel first. This takes about a half an hours, go get some lunch while it runs:

```
$ wget
https://github.com/bazelbuild/bazel/releases
/download/0.5.4/bazel-0.5.4-dist.zip
$ unzip -d bazel bazel-0.5.4-dist.zip
$ cd bazel
$ sudo ./compile.sh
```

Now, Java will run out of heap here, so we need to do the following modifications:

```
$ sudo vi scripts/bootstrap/compile.sh
```

Find the line with "run" on it, and add some memory flags, change it to the following:

```
run "${JAVAC}" -J-Xms256m -J-Xmx384m -
classpath "${classpath}" -sourcepath
"${sourcepath}"
```

Then, compile again:

```
$ sudo ./compile.sh
$ sudo cp output/bazel /usr/local/bin/bazel
```

Download and configure TensorFlow

Now we can actually download and configure TensorFlow:

```
$ git clone --recurse-submodules
https://github.com/tensorflow/tensorflow.git
$ cd tensorflow
```

I couldn't get the latest version of TensorFlow to install, since it had BoringSSL C99 compile issues. To fix this, checkout version 1.4.0, and configure:

```
$ git checkout tags/v1.4.0
$ ./configure
```

Say no to most things, including OpenCL, as shown in Figure 1.

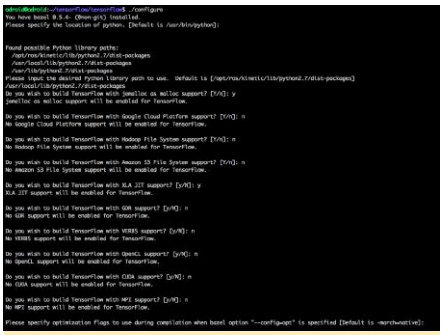


Figure 1 – Configuring TensorFlow

Build TensorFlow

Next, we need to build Tensorflow. If you thought Bazel took a long time to build, then you haven't built software before. Hold onto your hats, because we're in for a ride here:

```
$ bazel build -c opt --copt="-mfpu=neon-
vfpv4" --copt="-funsafe-math-optimizations"
--copt="-ftree-vectorize" --copt="-fomit-
frame-pointer" --local_resources
8192,8.0,1.0 --verbose_failures
tensorflow/tools/pip_package:build_pip_packa
ge
 Building…
 1,900 / 4,909 files… error.
```

Oops, NEON doesn't work. Ok, let's turn that off. But, we'll want to fix it later:

```
$ bazel build -c opt --copt="-funsafe-math-
optimizations" --copt="-ftree-vectorize" --
copt="-fomit-frame-pointer" --
local_resources 8192,8.0,1.0 --
verbose_failures
tensorflow/tools/pip_package:build_pip_packa
ge
 3,700 / 4,622 files… error.
 In file included from
tensorflow/compiler/xla/service/llvm_ir/llvm
_util.cc:30:0:
 ./tensorflow/core/lib/core/casts.h: In
instantiation of 'Dest
tensorflow::bit_cast(const Source&) [with
Dest = long long int; Source = void (*)
(const char*, long long int)]':
 tensorflow/compiler/xla/service/llvm_ir/llv
m_util.cc:400:67: required from here
 ./tensorflow/core/lib/core/casts.h:91:3:
error: static assertion failed: Sizes do not
match
```

In this case, XLA is causing problems. It's new, and not needed, so let's drop it for now and reconfigure and rebuild without it:

2,345 / 3,683 files...
3,112 / 3,683 files...
3,682 / 3,683 files...

Target //tensorflow/tools/pip_package:build_pip_package up-to-date:
bazel-bin/tensorflow/tools/pip_package/build_pip_package

Next, install it:

```
$ bazel-
bin/tensorflow/tools/pip_package/build_pip_p
ackage /tmp/tensorflow_pkg
$ sudo pip2 install
/tmp/tensorflow_pkg/tensorflow-1.4.0-cp27-
cp27mu-linux_armv7l.whl --upgrade --ignore-
installed
```

At first when I ran the following command, it ran using python 3 and failed to install, so after some googling and learning about pip filename rules, I figured it out and just used pip2 instead:

```
$ sudo pip install
/tmp/tensorflow_pkg/tensorflow-1.4.0-cp27-
cp27mu-linux_armv7l.whl --upgrade --ignore-
installed
```

Then, the most fun issue was when I first ran "import tensorflow", when I got this message:

```
>>> import tensorflow
 Traceback (most recent call last):
 File "", line 1, in
 File "tensorflow/__init__.py", line 24, in
 from tensorflow.python import *
 File "tensorflow/python/__init__.py", line
49, in
 from tensorflow.python import
pywrap_tensorflow
 File
"tensorflow/python/pywrap_tensorflow.py",
line 25, in
 from tensorflow.python.platform import
self_check
 ImportError: No module named platform
```

I googled the issue, and it seemed to be about locales, as shown at https://github.com/tensorflow/tensorflow/issues/36. So, I set a locale first, after also seeing that it needed to be capital US (http://bit.ly/2ifyle4), and rebuilt, and it still gave me the same issue:

```
$ export LC_ALL=en_US.UTF-8
$ export LANG=en_us.UTF-8
```

The next day, with fresh googling powers, revealed that actualy, it was just that I was just running it in the build directory! It has a directory called tensorflow in it, and python was looking up into that to find things. So just changing directories to another fixed the issue.

```
$ python2
 >>> import tensorflow
 >>> print(tensorflow.__version__)
 1.4.0
```

It looks like everything is working, so onto YOLOing.

Running YOLO

I'm sure there are a few implementations of YOLO out there by now, so let's pick one from https://github.com/experiencor/basic-yolo-keras:

```
$ git clone
https://github.com/experiencor/basic-yolo-
keras.git
$ cd basic-yolo-keras
```

Get weights from https://1drv.ms/f/s!ApLdDEW3ut5fec2OzK4S4RpT-SU, or raccoon from https://1drv.ms/f/s!ApLdDEW3ut5feoZAEUwmSMYdPlY

```
$ wget /tiny_yolo_features.h5
$ wget /tiny_yolo_raccoon.h5
```

Next, edit the configuration file, and change the model to "Tiny Yolo":

```
$ vi config.json
```

Download a picture of a racoon:

```
$ wget
https://upload.wikimedia.org/wikipedia/commo
ns/b/be/Racoon_in_Vancouver.jpg
```

Then, run the script:

```
$ python2 predict.py -c config.json -i
Racoon_in_Vancouver.jpg -w
tiny_yolo_raccoon.h5
```

It's missing the imgaug package, so we'll add it and its dependencies:

```
$ sudo pip2 install imgaug
$ sudo pip2 install keras
$ sudo pip2 install h5py
```

Both h5py and scipy take a little while to install. Can it find the raccoon in Figure 2?

Figure 2 – A test image of a raccoon

```
$ python2 predict.py -c config.json -i
Racoon_in_Vancouver.jpg -w
tiny_yolo_raccoon.h5
```

Yes, now that's a detected raccoon!

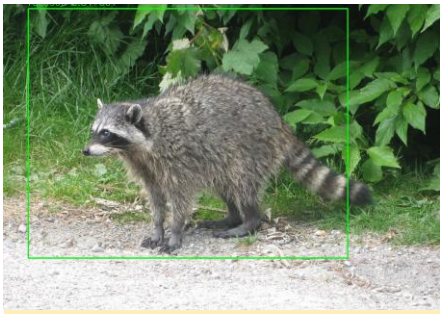


Figure 3 – YOLO has detected the image of the raccoon

For comments, questions and suggestions, please visit the original article at https://medium.com/@TomPJacobs/running-yolo-on-odroid-yolodroid-5a89481ec141.

Linux Gaming: Need for Speed II Second Edition

December 1, 2017 By Tobias Schaaf Gaming, Linux

In 1997, racing games were quite popular. One particular series that is well-known today was still quite new, though people were starting to know it and like it more and more. Need for Speed II Second Edition (NFS2SE) was released with 3DFX support, more tracks and cars, along with mirror mode and backward track mode, making it quite an improvement over the original Need for Speed II. In some countries, such as Germany, it was even included free with the purchase of a 3DFX accelerator card.

Figure 1 – Need For Speed II Second Edition

NFS2SE was a really good game by 1997 standards, as 3D graphics were just starting to get into people's homes and 3D racing games were still in their beginning stages. Having a decent 3D racing game with actual 3DFx Voodoo support was still rare at the time.

Along with the 3D graphics, this game had a lot to offer. One feature was full motion video (FMV) which could be seen in the intro as well as showcase videos for each car. There, you could see cars racing around different tracks and towns, or just out in the open. It offered lots of videos for the many cars that were featured in the game. Unlike modern Need for Speed titles, NFS2SE didn't have any storyline, so there was no story video to follow. Still, there was a good amount of video content as compared to other games of the time.

Figure 2 – Full motion video cut-scenes in NFS2SE

Figure 3 – Full motion video cut-scenes in NFS2SE

In fact, the game had quite a bit of extra content. The CD was packed with many pictures, videos, and information about the cars, and offered many different tracks and cars to play with.

Figure 4 – Select a showcase to see information, look at pictures, and watch videos for each car

Figure 5 – A video of the selected car from the showcase

Need for Speed II Second Edition on the ODROID

I recently found a open source project that aims to recreate the NFS2SE engine for modern systems using the 3D capabilities of current systems, including OpenGL ES 2.0 and SDL2, which would allow us to play on the ODROID. The re-creation orientates itself on the Glide (3DFx) version of the game, giving it a similar look to the original while allowing us to play on 1080P or other resolutions thanks to the scaling capabilities of SDL2.

So far the game looks good and seems to be fully playable. Even network multiplayer mode seems to work fine. I tested it on the ODROID-XU3 and the ODROID-U3, and it was running fine at full speed on both devices. Since the game runs in SDL2 with OpenGL ES 2.0, it is fully 3D accelerated and runs rather well, although I ran into some issues with speed if I ran the game in single thread mode.

Figure 6 – Starting a new race in NFS2SE

(Figure 6 – Starting a new race in NFS2SE)

The game offers different kinds of environment effects: rain drops, fog, and even bugs to obscure your view. Other areas offer nice ambient lighting effects.

Figure 7 – Entering a foggy rainforest

Figure 8 – Red glowing ambient light in a cave full of lava

Aside from that, NFS2SE for the ODROID offers joystick support and even uses force feedback (rumble support) so if you hit an object or a different road surface, the gamepad will rumble to emulate that sensation. I also tested the multiplayer mode on the two different ODROIDs and it worked well. I haven't tried splitscreen yet, but I am willing to bet that it will likely work also, meaning you will be able to easily play with a friend, or up to 8 players over the network.

Figure 9 – This landing will surely make the controller vibrate in your hand and trigger "force feedback"

So far, the only thing I've found that wasn't working correctly is the in-game menu. Here you can change volume to increase and decrease the sounds and music, but nothing else seems to works, not even continue or restart. For some reason I'm not able to select these menu points at all. I can only exit the menu using the ESC key. However, this shouldn't stop you from enjoying the game as is.

Figure 10 – The in-game menu seems to be partly broken as you can only get in and out using the ESC key

How to install the game

As usual, the game is available on my repository for Debian Jessie and Debian Stretch. Because the game is 32-bit only, ARM64 boards like the ODROID-C2 won't support the game. You can install it from my repository with:

```
$ apt-get install nfs2se-odroid
```

When you first run the game, you will need to install the game files from the original Need For Speed II SE CD. You will be asked to either point to a CD/Folder which includes the required folder (i.e. gamedata, fedata). If you're using my GameStation Turbo image, you can either plug-in a CD drive via USB and select the CD, or you can use CDEmu (Virtual CD) to mount most images formats. If you happen to have an .iso file you can select that instead and the setup will try to extract the files from there. After the files are copied, you should be able to go ahead and enjoy your game.

Figure 11 – Congratulations! You're now able to play Need For Speed II SE on your ODROID

Final Thoughts

Need For Speed II Second Edition might not be the best racing game out there, nor have top-notch technology and graphics by today's standard, but it's still a fun game to play, and the ability to race against each other over a network is something that isn't seen often on the ODROID. This game is well worthy of being part of the ODROID library. I hope you'll enjoy it just as much as I do.

Exploring Software-Defined Storage with GlusterFS on the ODROID-HC1: Part 2 – Client Performance

December 1, 2017 By Andy Yuen ODROID-HC1

In my previous article, I described how to setup a Distributed Replicated GlusterFS Volume as well as a simple Replicated Volume. I also described how to use the GlusterFS Native Client to access the volumes. In this part, I am going to show you how to setup NFS and Samba clients to access the GlusterFS volume and compare the performance of the different clients.

NFS Client

A NFS server is automatically set up when we install GlusterFS and create a Distributed Replicated volume. However, if your installation is like mine, when you execute the following command, you will find that the NFS servers are all offline, as shown in Figure 1.

```
$ gluster volume status
```

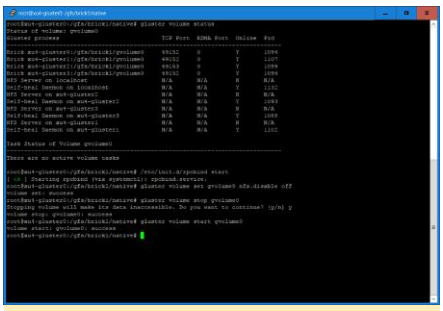


Figure 1 -nfs-offline

It is likely due to rpcbind not running, as evident in the /var/log/glusterfs/nfs.log shown in Figure 2.

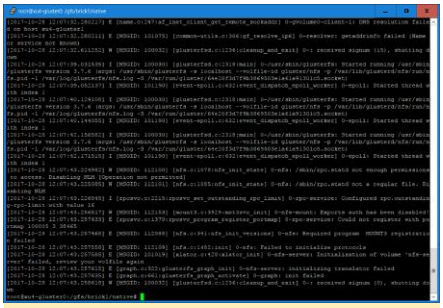


Figure 2 – rpc-error

Executing the following commands on all GlusterFS servers will start the NFS servers, where gvolume0 is the GlusterFS Distributed Replicated volume I created in Part 1:

```
$ sudo /etc/init.d/rpcbind start
$ sudo gluster volume set gvolume0
nfs.disable off
$ sudo gluster volume stop gvolume0
$ sudo gluster volume start gvolume0
$ sudo gluster volume status gvolume0
```

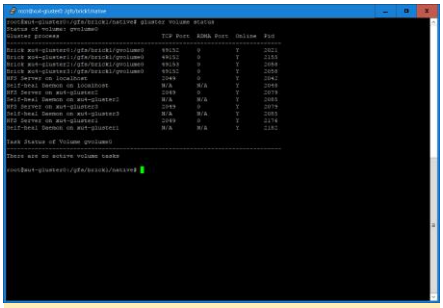


Figure 3 – nfs-online-all

I then setup a NFS client on one of the machines on my ODROID-MC1 namely, xu4-master,where xu4-gluster0 is one of the GlusterFS servers:

```
$ sudo apt-get update
$ sudo apt-get install nfs-common
$ sudo mkdir /mnt/nfs
$ sudo mount -t nfs -o vers=3,mountproto=tcp
xu4-gluster0:/gvolume0 /mnt/nfs
```

Now, you can access the GlusterFS volume on xu4-master. You can issue the following command on the list of 50 files we created in Part 1:

```
$ ls /mnt/nfs/testdir
```

SAMBA Client

The simplest way to access a GlusterFS volume is to export the Gluster mount point as the samba export and mount it

using CIFS protocol. Of course, you need to install the Samba and CIFS packages first:

```
$ sudo apt-get update
$ sudo apt-get install samba smbfs cifs
```

Then, you have to set up a password for samba,

```
$ sudo smbpasswd -a odroid
```

Edit the /etc/samba/smb.conf file with the following settings:

```
[global]
 security = user
 #guest account = nobody
 [gvolume0]
 guest ok = yes
 path = /mnt/gfs
 read only = no
 valid users = odroid
 admin users = root
```

Next, restart Samba and mount the share:

```
$ sudo /etc/init.d/samba restart
$ sudo mount -t cifs -
ouser=odroid,password=odroid
//192.168.1.80/gvolume0 /mnt/samba
```

To mount on the NFS share instead of Gluster share, simply unmount Samba, modify the /etc/samba/smb.conf file, restart Samba and mount the file system as follows:

```
[global]
 security = user
 #guest account = nobody
 [gvolume0]
 guest ok = yes
 path = /mnt/nfs
 read only = no
 valid users = odroid
```

Restart samba and mount the share:

```
$ sudo /etc/init.d/samba restart
$ sudo mount -t cifs -
ouser=odroid,password=odroid
//192.168.1.80/gvolume0 /mnt/samba
```

Client Performance

To compare the performance of the various clients, we need a baseline and the baseline is the native file system performance, i.e., a partition mounted locally on a server. This implies that we have 5 different clients to compare. They include:

1. Native File System – performance test is running on a server where a local disk partition is mounted
2. GlusterFS Native Client – mounted on a machine which is not a GlusterFS server using the GlusterFS Native Client
3. Gluster NFS client – mounted on a machine which is not a GlusterFS server using the GlusterFS NFS Client
4. Samba client based on GlusterFS Native Client – CIFS share of the GlusterFS Native Client file system
5. Samba client based on GlusterFS NFS Client – CIFS share of the GlusterFS NFS Client file system

The file system performance benchmark tool used is iozone, which is not in the Ubuntu software repository, but can be downloaded from http://bit.ly/2BnxxfA. It generates and measures a variety of file operations. We are using the following 4 benchmarks to compare client performances: Single-Thread Write, 8-Thread Write, Single-Thread Read, 8-Thread Read.

Single-Thread Write

The iozone command used in the test is:

```
$ iozone -w -c -e -i 0 -+n -C -r 64k -s 1g -
t 1 -F path/f0.ioz
```

The options used are:
-c Include close()
-e Include flush in time calculations. (-c -e options are used together to measure the time it takes for data to reach persistent storage)
-w Do not unlink temporary files when finished using them
-i 0=write, 1=read (we only used 0 and 1 in these tests
-+n Save time by skipping re-read and re-write tests
-C Show how much each thread participated in the test
-r data transfer size
-s per thread file size
-t number of threads
-F List of files

The command is run for every client. The output of the command is shown in Figure 4, and the result is summarized in the bar chart in Figure 5.

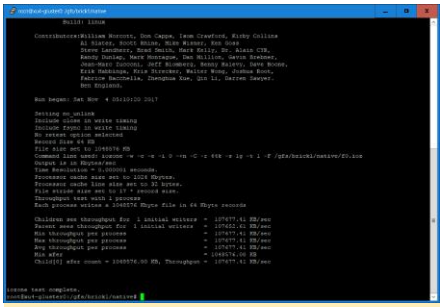


Figure 4 – native-w1

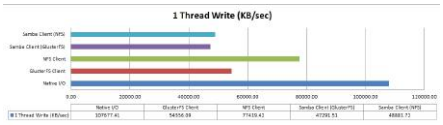


Figure 5 – 1ThreadWrite

Native is the fastest, followed by NFS and GlusterFS native clients. As expected, the Samba clients are the slowest in our configuration because they rely on the underlying GlusterFS native and NFS clients.

8-Thread Write

The command used is:

```
$ iozone -w -c -e -i 0 -+n -C -r 64k -s 1g -
t 8 -F path/f{0,1,2,3,4,5,6,7,8}.ioz
```

The output is shown in Figure 6, and the result graphics are shown in Figure 7.

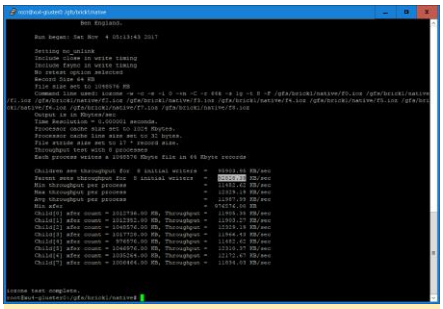


Figure 6 – native-w8

Figure 7 – 8ThreadWrite

In the multithreading write benchmark, the Native result is the fastest, followed by the NFS and Samba client using NFS. Note that the Samba client using NFS is faster than the NFS client itself, for which I do not have a clear explanation.

Single-Thread Read

The cache is cleared before the read benchmark:

```
$ sync
$ echo 1 > /proc/sys/vm/drop_caches
$ iozone -w -c -e -i 1 -+n -C -r 64k -s 1g -
t 1 -F path/f0.ioz
```

The resultant results' graphic are shown in Figure 8.

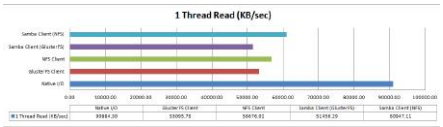


Figure 8 – 1ThreadRead

The result is consistent in that the Native result is the fastest, followed by NFS-based clients and then GlusterFS native client-based clients. Again, I have no explanation as to why the Samba client using NFS is faster than the NFS client itself.

8-thread Read

Finally, run the multithread read benchmark using the following commands:

```
$ sync
$ echo 1 > /proc/sys/vm/drop_caches
$ iozone -w -c -e -i 1 -+n -C -r 64k -s 1g -
t 8 -F path/f{0,1,2,3,4,5,6,7,8}.ioz
```

The result is shown in Figure 9.

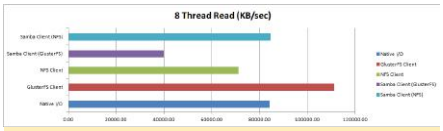


Figure 9 – 8ThreadRead

The performance is quite different here, since the GlusterFS native client is the fastest, likely due to distribution. Files are stored on different servers, which adds parallelism in retrieval, unlike writes, which adds overhead in writing data to multiple servers.

Auto-Failover and high availability clients

Of all the clients tested, only the GlusterFS Native Client provides auto-failover and high availability capability. This means that if the GlusterFS server that specified in the mount command fails, it will automatically switch over to use another Gluster server in our Replicated or Distributed Replicated Volume.

The NFS and Samba clients used do not have such capabilities. If you want that capability for NFS, you have to disable the GlusterFS NFS server and install the NFS-Ganesha server (http://bit.ly/2BuH9Ek).

Similarly, for Samba/CIFS, you have to install the Samba VFS Plugin from http://bit.ly/2i7gMjI. In addition to providing high availability and auto-failover, it also uses libgfapi to avoid the performance penalty to go between user and kernel mode which happens in our Samba client based on GlusterFS Native Client used in our test.

Note that the plugin for gluster is absent in the Ubuntu package samba-vfs-modules. I encourage you to look into them if you want high availability and auto-failover for your NFS and Samba clients.

Conclusion

I have shown you how to setup GlusterFS Replicated and Distributed Replicated Volumes using ODROID-HC1 devices, and how to access them using GlusterFS Native Client, NFS and Samba clients. I have also shown you their performance in charts for easy comparison. You now have

sufficient information to select the the appropriate client. Personally, I think it is a good enterprise technology that lends itself to easy home use. ODROID-HC1s are more economical and flexible than off-the-shelf NAS systems, in my opinion. I hope you will share my enthusiasm in using them at home.

Meet An ODROIDian: Andrea Cole, Assistant Editor of ODROID Magazine

December 1, 2017 By Meet an ODROIDian

Please tell us a little about yourself.
I'm currently a sales admin for Lab Manager, an industry-focused publication for the scientific community. I've been a part of their parent company, LabX Media Group, for over 10 years, and have been working specifically in the Lab Manager division for four years. I live in Canada, a couple hours north of Toronto, Ontario, in a small town situated on the shores of Georgian Bay.

I have a bachelor of arts degree in Sociology from Laurentian University. I had originally started out pursuing a degree in Psychology, but after taking a few sociology classes and realizing that I was enthusiastically pouring all my energy into these classes and neglecting my psych classes, I decided switching majors was probably a smart move.

I currently live with my two daughters who are still in high school, and my partner, who had previously worked in the IT industry, but now keeps to more horticultural work since, as he puts it, "working in the industry was ruining a perfectly good hobby." He's the one that got me interested in electronics.

Figure 1 – Andrea enjoys spending time with her two daughters

How did you get started with computers?
As a kid, my family had the occasional home computer, but it was mostly a work computer used for word processing, so I didn't take to much of an interest in it. When the Internet became more popular, I had a lot more exposure to computers, but mainly used them as a tool for accessing the Internet. I spent a lot of time on a few online forums, and as a budding musician I was a frequent visitor to the On-Line Guitar Archive (OLGA). However, it wasn't until my current job that I began to pick up some basic programming knowledge. Since part of my job description involves email marketing, I have picked up enough HTML and CSS to do some basic web design. For a long time I was working with a web crawler, and from that I developed a basic understanding of regular expressions. Overall, I'm still pretty new to this, suffice it to say.

How do you use your ODROIDs?
I have been using the ODROID-C2s mostly to replace the older C1 and Raspberry Pi, in our entertainment setup that has grown into almost every room in the house, thanks to the ODROIDS and a partner that can't be trusted not to add extra functionality to almost everything in the house. For example, I came home one day to discover that my 1960s cabinet record player now has WiFi, a web interface, and the ability to stream music from the Internet.

Which ODROID is your favorite and why?
I can't say I've really thought about my favorite, as I've really only had experience with the ODROID-C1 and ODROID-C2, and limited experience with the ODROID-XU4. Given that my next Hardkernel order will likely contain at least a half-dozen ODROID-C2s, I'd probably go with that. We've got more possible uses for the ODROID-C2 than any other boards currently available.

Whom do you admire in the world of technology and why?
Is Tony Stark real? No? Dang. [Editor's Note: Elon Musk is as close as we have to a real Tony Stark]

What benefits do you see in helping others learn more about ODROIDs?
Single-board computing using resources such as the ODROID has the benefit of helping people understand the technologies that are becoming more and more embedded in their everyday lives. It's important that people learn about what powers their tech, so they can de-mystify the magic little boxes in their lives and empower

themselves to create their own devices that will do what they want, how they want, without being limited by profit-driven corporate ideals.

What hobbies and interests do you have apart from computers?

I've had an avid interest in music since my early teen years. I also have a capacity for an almost infinite amount of useless pop culture trivia. 10 years ago, I took up painting as a hobby, and in recent years I've put an increased effort into honing that craft. My artwork can be found at https://andrea-cole.pixels.com/.

Figure 2 – Artwork by Andrea called A Tragically Hip Mountain Goat"

Figure 3 – Artwork by Andrea called Victoria Harbour Town Dock"

What advice do you have for others who wish to learn more about computers?

Look into online programming courses that you can take in your spare time. Some of these offerings are more on-the-ball than others, but if you know people who are already knowledgeable on the subject, they may be able to provide guidance towards reputable sites for online learning. Try not to be overwhelmed by the seemingly immense amount of information. After that, it's a matter of making a commitment to carve out the time for yourself and just do it.