SWI Prolog • ODROID-VU • Linux Gaming • Adafruit • Cryptomining

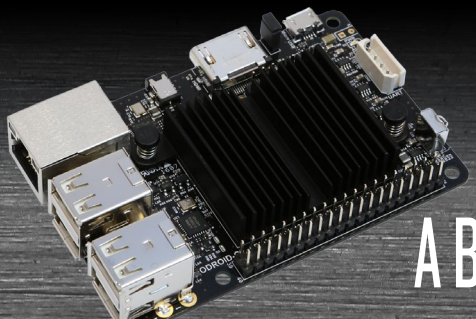
ODROID Magazine

Year Four
Issue #44
Aug 2017

Experience the future with your ODROID:

Artificial *Intelligence*

Migrating From Ubuntu Mate to Lubuntu



Android on the ODROID-C2: A Beginner's Guide

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

Pollin ELECTRONIC

We are now shipping the ODROID-C2 and ODROID-XU4 devices 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

EDITORIAL

Artificial intelligence is one of the most exciting computing frontiers of the 21st century. Technology such as Amazon Alexa, Google Assistant, and Microsoft Cortana are compact, affordable and useful personal assistants, but they all use closed-source proprietary code, which raises privacy concerns. A solution for ensuring that the user has complete control over the information that is shared and stored is to use open-source applications such as Mycroft and Prolog. Compiling the code yourself and installing it on an ODROID is an easy, inexpensive way to build a personal assistant to fit your exact needs while also protecting your privacy.

Another emerging application of advanced technology is cryptocurrency mining. Hardkernel engineers built a ODROID-XU4 demonstration cluster to show the stability of Kernel 4.9 and calculate the financial viability of long-term mining. Tobias discusses whether he is a Nintendo fanboy, Jörg shows us how to control the display backlight in Android, Miltiadis helps us migrate from Ubuntu MATE to Lubuntu, Dennis demonstrates a project using the Adafruit OLED display, and we present a guide to getting started with Android on the ODROID-C2.

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/1ypImXs.
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

OUR AMAZING ODROIDIAN STAFF:

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 prebuilt 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

Is still taking care of the dog, and yes, he did get a Playstation 4. As for the PDF version of ODROID Magazine, it was a great 44 issue journey. Onward to the new frontier of the HTML interactive version!

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 anODROID-U2, a number of ODROID-U3's, and Xu4'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.nicolecscott.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.

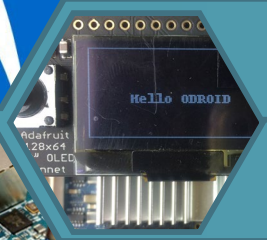
Venkat Bommakanti, Assistant Editor

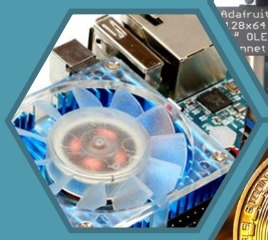
I'm a computer enthusiast from the San Francisco Bay Area in California. I try to incorporate many of my interests into single board computer projects, such as hardware tinkering, metal and woodworking, reusing salvaged materials, software development, and creating audiophile music recordings. I enjoy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

Andrea Cole, Assistant Editor

I live in Ontario, Canada, and while relatively new to the ODROID, I have ten years of experience in online publishing under my belt. I currently have has single-board computers--programmed by my partner--running our media center, a game emulator, and my music library. In my spare time, I'm an artist, writer, and musician. I plan to incorporate some of the knowledge I gain into some interactive, mixed-media artwork.

INDEX





MYCROFT

OPEN SOURCE ARTIFICIAL INTELLIGENCE

by Adrian Popa

Ever since KITT made its appearance as a talking car back in the 1980s, everybody dreamed about the day when they would have their own personal talking assistant. The good news is that we are almost there. The bad news is that there is a high cost associated with the lack of privacy and data mining done by big companies. It would be nice to have a voice-controlled assistant that does not share its data with the cloud. Unfortunately, we are probably still 10 years away from that point, but we are making progress in the right direction. In this article, we are going to install an open-source artificial intelligence platform that runs on your ODROID called Mycroft (`https://mycroft.ai/`).

Mycroft consists of several interconnected components that handle speech acquisition, keyword recognition (done locally), speech to text translation (currently done through a Google service), and skills engine and text to speech (done locally by mimic). Mycroft communicates with several entities on the Internet, using its own cloud to store and retrieve settings and API keys for various services and incorporating Google speech-to-text services. In addition to this, the skills engine may communicate with online services to retrieve data for the end user.

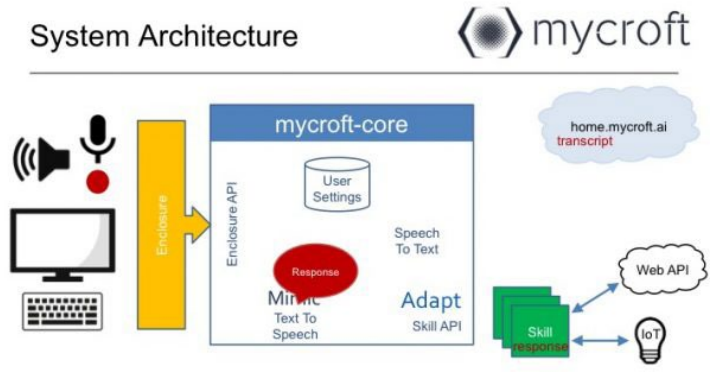


Figure 1 - Mycroft architecture

From the start, people will compare Mycroft with other personal assistants such as Amazon Echo, Alexa, and Google Assistant. However, they are quite different. Mycroft is built to be open and non-intrusive, while all the other assistants are in constant contact with your profile data and can respond to requests involving your personal data, such as your contact list or personal calendar. By default, Mycroft does not have access to that information and is mostly used to query public information sources, such as the weather, Wikipedia and Wolfram Alpha (`https://www.wolframalpha.com/`). Some may think this is limiting, but if you value your privacy, then you know it is not. A lot of advanced functionality can be enabled by the end user by activating certain skills, such as Google account integration. In terms of AI complexity, Mycroft is simpler than the other AI voice assistants, which run completely in the cloud and benefit from complex recurrent neural networks.

One major privacy concern is having a visible or hidden device constantly listening to conversation. Analyzing what data is leaked about you by your personal assistant should be high on everyone's priority list (`http://bit.ly/2gY9qKG`). The open-source advantage of Mycroft is that, the network queries are logged and easily consulted by the average user, so you can actually see what the device is actually doing behind the interface.

Mycroft could track what you speak in their cloud, but network logs show that they do not save a transcript of what you speak. The speech data is sent to Google servers to be con-

verted to text, because speech to text is still a difficult problem which generally requires the use of neural networks (http://bit.ly/2eGLJJz). Theoretically, Google can still track what you are saying, and can link it to your IP address, which compromises privacy. The Mycroft team has started work on an open speech-to-text engine to solve this issue, but it will realistically take a long time before this can be run locally (https://openstt.org/).

So, If you want a voice activated personal assistant, and are comfortable with the security implications of having one, let's see how we can get Mycroft running on the ODROID.

Installation

If you have an ODROID-U3, you can skip everything and install a Mycroft-enabled Debian image created by forum user @nold, available at http://bit.ly/2tT2Hq1. However, if you do not want to dedicate an ODROID for a single task, you can install Mycroft on a general-purpose ODROID-C2 which runs the Hardkernel-provided Ubuntu 16.04 image. You can use the same steps to install on a different ODROID model, such as C1 or XU4.

The first thing you are going to need, independent of your ODROID model, is swap space. The installation process needs to compile the text-to-speech component called Mimic, and this takes up a lot of memory. You should also do a clean reboot to free up any used memory in case your ODROID has been up for a long time.

To create a temporary 4GB swap file on your SD/EMMC run the following commands:

```
$ sudo dd if=/dev/zero of=/swapfile bs=1M count=4096
$ sudo mkswap /swapfile
$ sudo swapon /swapfile
```

You can test that swap has been activated with the following command:

```
$ free -m | grep Swap
```

To do the actual installation, you may want to read the documentation located at https://docs.mycroft.ai/. We will be going with the git clone install method (http://bit.ly/2eGAp05). Mycroft will be installed inside a python virtual environment inside your user home directory, typically /home/odroid. It will also run as a regular user. For the installation and compilation steps, you will need about 2.5 GB of free space.

```
$ cd ~
$ sudo apt-get install git
$ sudo git clone https://github.com/MycroftAI/my-
croft-core.git
```

```
$ cd mycroft-core/
$ sudo ./build_host_setup_debian.sh
$ sudo chown -R `whoami`:`whoami` .
$ ./dev_setup.sh
```

The dev_setup.sh step will create the virtual environment, set up the Python modules needed and then will compile Mimic. After Mimic has been compiled, which takes about three and a half hours using an ODROID-C2 @1.75GHz, you may be prompted with this error while compiling pygtk:

```
configure: error: cannot guess build type; you must
specify one
make: *** No targets specified and no makefile found.
Stop.
make: *** No rule to make target 'install'.  Stop.
```

This is an inconsequential error and can safely be ignored. It indicates that there is a desktop integration component that has failed to build, but we are not going to use it. In the future, if you want to update Mycroft, you can redo the installation steps (especially the git clone step), but run dev_setup.sh -sm to skip the Mimic build step.

If the build went well, we can do some cleanup and free up some disk space. The following commands turn off swap and reclaim 4 GB of disk space:

```
$ sudo swapoff /swapfile
$ sudo rm -f /swapfile
```

We can also cleanup about 500 MB worth of mimic temporary files by running the following commands:

```
$ cd mimic
$ make clean
```

Unfortunately, Mimic is installed within its source directories, so it's difficult to separate it and clean it up further.

Startup

Mycroft does not come with a systemd startup unit, but we can create one around its startup script. Mycroft's startup script is located in ~/mycroft-core/mycroft.sh. It accepts parameters such as start and stop, and manages 4 screen instances, which each run a key Mycroft component. To create a systemd startup unit, run the following commands, making sure to adjust the username and paths if you are not running as the "odroid" user:

```
$ sudo vi /etc/systemd/system/mycroft.service
```

```
[Unit]
Description=Mycroft personal AI
After=pulseaudio.service

[Service]
User=odroid
WorkingDirectory=/home/odroid/mycroft-core
ExecStart=/home/odroid/mycroft-core/mycroft.sh start
ExecStop=/home/odroid/mycroft-core/mycroft.sh stop
Type=forking
Restart=always
RestartSec=3

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

To start up Mycroft on every boot, and to manually start it up now, type the following commands:

```
$ sudo systemctl enable mycroft
$ sudo systemctl start mycroft
```

If the default sound settings are working, you should see Mycroft complaining that it is not paired with its cloud, and it should offer a pairing code. However, let us assume that you do not have sound yet, or maybe you did not understand what Mycroft indicated. We will need to get accustomed to the backend processes.

Backend and debugging

When Mycroft starts up, it creates 4 screen instances, where it runs processes for voice, skills, command-line interface, and services. You can list and connect to these screen instances with the following command:

```
$ screen -list
There are screens on:
      5705.mycroft-cli       (07/13/17 13:50:28)
(Detached)
      5690.mycroft-voice     (07/13/17 13:50:28)
(Detached)
      5675.mycroft-skills    (07/13/17 13:50:28)
(Detached)
      5640.mycroft-service   (07/13/17 13:50:28)
(Detached)
```

If you connect to the mycroft-voice screen, you will be able to see if there are any errors with audio input or output (such as - there is no default audio output device). To connect and interact with a screen session you can run the following command:

```
$ screen -r 5690.mycroft-voice
```

Note that the number is the process PID and will change on Mycroft restart. You should identify it with screen -list, as shown above. To detach from a screen session without terminating the screen process, use the key combination CTRL+A D. See the screen manual at `http://bit.ly/2tsC4zF` for more key bindings.

Regular text logs from these services are also saved in ~/mycroft-core/scripts/logs/ and can be consulted with regular tools such as “grep” or “tail -f”.

Mycroft also comes with a debugging mode where it starts only the engine, skills, and CLI components, so you can debug various problems. To start Mycroft in debug mode, type:

```
$ ~/mycroft-core/mycroft.sh start -d
```

Debug view

Configuration

Mycroft’s primary configuration file is located at ~/mycroft-core/mycroft/configuration/mycroft.conf. You should not edit this, but instead make a copy at ~/.mycroft/mycroft.conf, to avoid the settings being overridden on Mycroft upgrade:

```
$ cp ~/mycroft-core/mycroft/configuration/mycroft.conf
~/.mycroft/
```

The configuration file contains general settings, such as the language (I tested using English), units, location, as well as specific skill parameters for default skills. These will likely move to different configuration files in the future. At this point you can edit the file at ~/.mycroft/mycroft.conf and set your preferred settings, including location, so that the answers are relevant to you. For example, if you leave the default location of Lawrence, Kansas, when you ask for the time or weather, you will get the time and weather from Lawrence, Kansas. Some of these settings can also be set from the Mycroft cloud after pairing the device.

Audio

If you are running a desktop image with sound output via HDMI and a microphone, it is like that sound will work fine without manual intervention. However, if you are running a server image without a desktop environment, or if you have multiple microphones/sound outputs, you may want to manually configure Mycroft to use a specific sound device.

Although Mycroft supports both ALSA and PulseAudio, we are going to use PulseAudio as a sound backend, because it's more flexible. For instance, it's easier for multiple processes to use the microphone at the same time.

Usually, PulseAudio runs as a user process and starts when you log in to your desktop environment. This is not compatible with having Mycroft start as a service at startup, since there would be no PulseAudio to connect to. We need to run PulseAudio as a system service instead. More details are available at http://bit.ly/2vzTOnj.

```
$ sudo apt-get install pulseaudio
$ sudo vi /etc/systemd/system/pulseaudio.service
[Unit]
Description=PulseAudio Daemon

[Install]
WantedBy=multi-user.target

[Service]
Type=simple
PrivateTmp=true
ExecStart=/usr/bin/pulseaudio --system --real time
--disallow-exit --no-cpu-limit
```

In order to allow user processes to communicate with a root PulseAudio, we need to edit the configuration located at /etc/pulse/system.pa. You will need to add the following lines to the configuration:

```
#allow localhost connections
load-module module-native-protocol-tcp auth-ip-
acl=127.0.0.1
```

Before we run PulseAudio as a service, we need to identify the microphone (input) and the speakers (sink) that we want to use. For this, we will need to run PulseAudio as a user:

```
$ pulseaudio -D
```

Next, we can get a list of output devices (sinks) that Mycroft can play to. The idea is that if your ODROID has both HDMI connected and a USB sound card, you can have Mycroft use the soundcard for output, so that you can hear it even when the TV is off.

```
$ pacmd list-sinks | egrep "index|name:"
   index: 0
        name: <alsa_output.usb-0d8c_C-Media_USB_Head-
phone_Set-00.analog-stereo>
  * index: 1
        name: <alsa_output.platform-odroid_hdmi.
analog-stereo>
```

In the output above you can see that the HDMI output is the default sound output. We will want to tell Mycroft to use card 0 (or better, we can index it by name) for sound. To do this, edit ~/.mycroft/mycroft.conf and change the following line:

```
play_wav_cmdline": "aplay %1
```

to

```
play_wav_cmdline": "paplay -d alsa_output.usb-0d8c_C-
Media_USB_Headphone_Set-00.analog-stereo %1
```

This change will route all voice messages generated by Mycroft to the USB soundcard.

Some skills will play podcasts, or mp3 content like the news, so if you want to hear that over a different sound output, you need to change the following setting as well. Change the following line:

```
play_mp3_cmdline": "mpg123 %1
```

to

```
play_mp3_cmdline": "mplayer -ao pulse::alsa_output.
usb-0d8c_C-Media_USB_Headphone_Set-00.analog-stereo
%1
```

The issue is that mpg123 does not have a switch for selecting a PulseAudio sink, so we need to exchange it with mplayer, which needs to be installed in your system:

```
$ sudo apt-get install mplayer
```

One last thing to do is to set a default microphone. In my case, I have three microphone inputs: one from the sound card (not connected), one from my webcam, and one from HDMI.

```
$ pacmd list-sources | egrep "index|name:"
    index: 1
        name: <alsa_input.usb-Sonix_Technology_Co.__
```

```
Ltd._USB_2.0_Camera-02.analog-mono>
    index: 2
        name: <alsa_output.usb-0d8c_C-Media_USB_Head-
phone_Set-00.analog-stereo.monitor>
  * index: 3
        name: <alsa_input.usb-0d8c_C-Media_USB_Head-
phone_Set-00.analog-mono>
    index: 4
        name: <alsa_output.platform-odroid_hdmi.
analog-stereo.monitor>
    index: 5
        name: <alsa_input.platform-odroid_hdmi.
analog-stereo>
```

I want to use my webcam's microphone for Mycroft input, so I will set it as the default option in /etc/pulse/system.pa:

```
#set default microphone
set-default-source alsa_input.usb-Sonix_Technology_
Co.__Ltd._USB_2.0_Camera-02.analog-mono
```

Save the file, and you are ready to start the service:

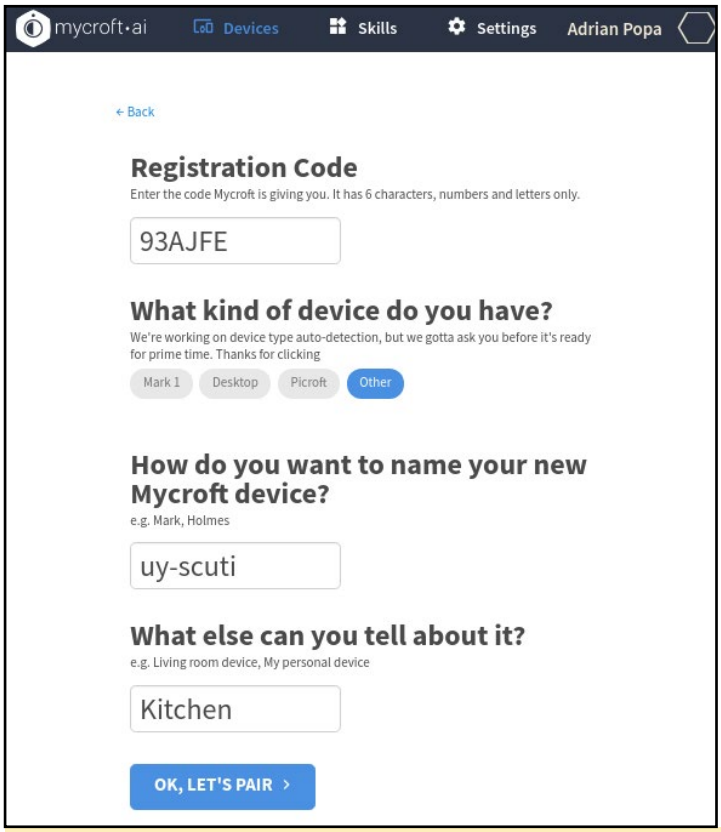
```
$ sudo systemctl enable pulseaudio
$ killall pulseaudio
$ sudo systemctl start pulseaudio
```

Pairing and interacting with Mycroft

Right now we can restart the whole system, and Mycroft should start after PulseAudio has started. The microphone should work as expected and so should audio output. You are now ready to register Mycroft. To have Mycroft process your speech, you should start your questions with "Hey Mycroft!". To get a pairing code, ask "Mycroft, register my device". Mycroft should then speak its registration code which usually consists of 6 characters. If you cannot hear the code or cannot understand it, you can find it in the mycroft-skills.log located at ~/mycroft-core/scripts/logs. You will need to register at http://home.mycroft.ai and navigate to Devices -> Add device.

You should now be ready to use Mycroft. You can go ahead and ask it for various information, such as:

Hey Mycroft, what's the time?
Hey Mycroft, what day is today?
Hey Mycroft, wiki European Union.
Hey Mycroft, tell me about Abraham Lincoln.
Hey Mycroft, tell me a joke.
Hey Mycroft, why is 6 afraid of 7?
Hey Mycroft, sing me a song!
Hey Mycroft, what is 2+5?



Registering online

Hey Mycroft, tell me the news.
Hey Mycroft, set a reminder for 5 minutes.
Hey Mycroft, what's the weather?
Hey Mycroft, what's the forecast for tomorrow?

To know what you can ask for, consult the basic skills documentation page at `http://bit.ly/2uourGy`. You can also review the mycroft-skills.log file and can see what keywords are registered when a skill is loaded. For example, the joke skill registers the following key phrases: joke, make me laugh, brighten my day, tell me joke.

It is all about the skills

The basic skills of Mycroft are quite useful, but you can do so much more with it by using third party skills, available from the official repository at `http://bit.ly/2tsEYh1`. Keep in mind that the quality of the skills may vary depending on the implementation.

In the example below, we will be adding support for diagnostics, such as having Mycroft tell you CPU usage, free space, and Home Assistant integration, so that Mycroft can read values from Home Assistant or control switches with your voice.

Third party skills can be installed in /opt/mycroft/skills, and will be automatically loaded on Mycroft restart. To add the diagnostics skill, simply clone it from its GitHub link:

```
$ cd /opt/mycroft/skills
```

```
$ git clone http://bit.ly/2uOP4N1
```

The skill can be configured to run a user-defined script and read its output. The path to the user-defined script needs to be configured inside ~/.mycroft/mycroft.conf. Note that the location of third-party configuration is bound to change in new Mycroft releases, so consult the official documentation for the correct location and syntax.

```
"DiagnosticSkill": {
   "script": "/home/odroid/bin/usbreset.sh"
 }
```

Next, restart Mycroft:

```
$ sudo service mycroft restart
```

After the service re-launches, you will be able to use queries such as "Hey Mycroft, what is the current CPU usage percent?", "Hey Mycroft run diagnostics", which runs your custom script, or "Hey Mycroft, what's your uptime?"

Similarly, we can install the Home Assistant skill and integrate Mycroft with a running Home Assistant instance. This skill has some external dependencies which need to be installed as well:

```
$ cd /opt/mycroft/skills
$ git clone http://bit.ly/2gVOojm
$ workon mycroft
$ cd HomeAssistantSkill
$ pip install -r requirements.txt
```

Also, you will need to edit ~/.mycroft/mycroft.conf and add the following lines:

```
"HomeAssistantSkill": {
  "host": "hass.mylan.net",
  "password": "mysupersecrethasspass",
  "ssl": true|false
}
```

After restarting Mycroft, you can now query the state of your Home Assistant entities by asking things like "Hey Mycroft, turn on air conditioning power" to activate the "Air Conditioning Power" switch, or "Hey Mycroft, ask home assistant what is kids room temperature?" to query a sensor called "Kids room temperature". The skill will try to match what you say against the names of Home Assistant's entities so that you can control Home Assistant with your voice! I personally think this is the coolest skill! For further skill examples, check out the YouTube video at http://bit.ly/2vfh90H.

Mycroft's future

Mycroft started as a simple AI, but with the support of the community, it is evolving into something bigger. For now it works in a question/answer pattern, without keeping the state of a conversation, but work is being done to change that in the future, so you will eventually be able to have a conversation with Mycroft (http://bit.ly/2w7L6MH). In terms of performance and used resources, since it was designed to run on a Raspberry Pi, it has no problem running on ODROIDs, even on a C1). On my ODROID-C2, with other background processes running, Mycroft uses about 20% CPU on 2 cores with the governor keeping the CPU frequency at 500 MHz when idle.

Currently, Mycroft might be at best mildly useful and not really a personal assistant, but it has potential. You could compare it to MS-DOS back in the 1980s, which was clunky, but got the job done. The concern is that all personal assistants make mistakes, and sometimes with funny consequences (http://bit.ly/2uXQTbg). I think that in 10 to 20 years, the personal assistant will evolve to something that is hard to predict, like Skynet in the Terminator movies, but I hope that the personal assistant of the future will be community-based and open-sourced. In the event that you get stuck or have further questions, please consult the support thread at http://bit.ly/2tt3crC or the Mycroft community forums at https://community.mycroft.ai/.

ARTIFICIAL INTELLIGENCE PROGRAMMING

USING PROLOG WITH AN ODROID-XU4

by Birmsoo Kim

Researchers in Artificial Intelligence (AI), have studied problem solving skills used by humans and implemented those skills on a machine. This research showed that people tend to infer solutions of a problem using facts and the relations between those facts. Prolog is a programming language that was invented in order to perform the inference necessary to solve problems involving facts and rules.

Installing Prolog

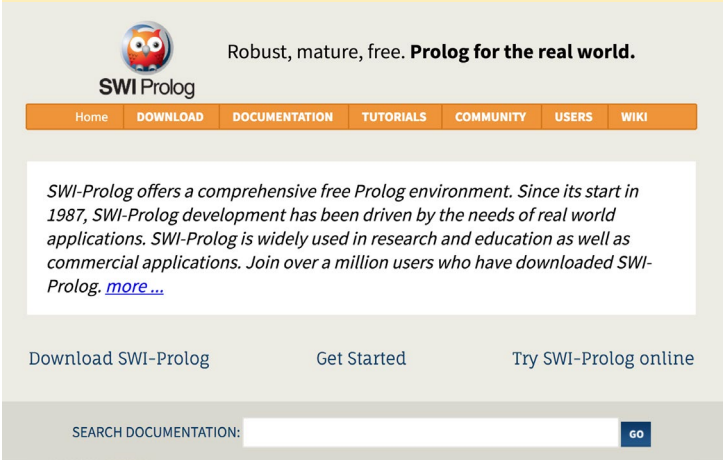
Recently, IBM announced that their AI system partially depends on the Prolog programming language. Using Prolog, we are going to build a simple intelligent machine on ODROID-XU4. If your XU4 is running Ubuntu, you can install SWI-Prolog on your machine with following commands:

```
$ sudo apt-get update
$ sudo apt-get install swi-prolog
```

If you want to install SWI-Prolog on your machine that has a different OS, such as Windows or MacOS, visit the website `www.swi-prolog.org`. You can get more information related to installation. Figure 1 shows the first page of SWI-Prolog site.

Once SWI-Prolog is installed on your system, it is time to get into AI programming.

Figure 1 - SWI-Prolog home page



Implementation of a Prolog program

Let us create a Prolog file named family.pl. This file describes the relationship of a family with facts and rules:

```
%fact
sex(lea, female).
sex(dan, male).
sex(jay, male).
sex(esther, female).
sex(irene, female).
sex(praise, female).
sex (william, male).

parent(lea, jay).
parent(dan, jay).
parent(dan, esther).
parent(jay, praise).
parent(jay, irene).
parent(lea, william).

%rules
offspring(Y,X) :- parent(X,Y).
mother(X, Y):-parent(X,Y),female(X).
father(X, Y):-parent(X,Y),male(X).
grandparent(X,Z):-parent(X,Y),parent(Y,Z).
sister(X,Y):-parent(Z,X),parent(Z,Y),female(X),diffe
rent(X,Y).
different(X,Y):-X=Y, !, fail.
different(_,_).
Predecessor(X,Z):-parent(X,Z).
Predecessor(X,Z):-parent(X,Y),predecessor(Y,Z).
```

The family.pl file consists of facts and rules. Facts show data that are always true. For example, sex(lea, female) is a

fact that is used to describe the gender of Lea as female. As you expect, sex(dan, male) is another fact that Dan is male. Other facts such as parent(lea, jay) and parent(dan, jay) show the relation of Jay to Lea and Dan. These facts represent that both of them are the parents of Jay.

Rules are also described in this file. The rule "grandparent(X, Z) :- parent(X, Y) ,parent(Y, Z)" describes the condition of being a grandparent of someone. The conclusion part is on the left side, and the condition part is on the right side.

This rule is represented by a condition as follows:

For all X and Z, X is grandparent of Z if

X is a parent of Y and

Y is a parent of Z. [1]

Two rules below are representing mother and father. To be the father of Y, the Y should be the parent and male:

mother(X, Y):-parent(X,Y), sex(X, female).

father(X,Y):-parent(X,Y), sex(X, male).

Figure 2 shows the relation of family in family.pl. The higher position is the parent of a lower position. Therefore, we

Figure 2 - Family Tree

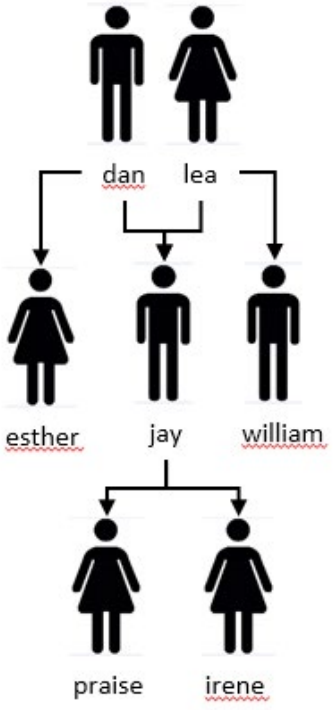


Figure 3 - swipl and upload

Figure 4 - Parent of Irene

Figure 5 - Grandparents

can find that the parent of Irene is Jay, and the parents of Jay are Dan and Lea.

Running a Prolog program

You can start the prolog interpreter with the "swipl" command:

```
$ swipl
```

When the family.pl file is ready, run consult(filename) or [file name] to load the family.pl file on the system. As shown in Figure 3, the system responded "yes" to show that the Prolog file was uploaded successfully.

```
?-consult(family)
?- [family]
```

Now, you can give a question to about the family relation. If you want to know who is the father of Irene, a parent(X, Irene) query will show the answer. X is variable that will match and return the name of irene's dad. The prolog interpreter search the given facts and finds jay as the return value of X as shown in Figure 4.

You can figure out who is a grandparent of Irene by grandparent(X, Irene). You will get the solution with facts and rules. The rule grandparent(X, Z) :- parent(X, Y) ,parent(Y, Z) will use facts parent(lea, jay), parent(dan, jay), and parent(jay, irene) as conditions. When the two conditions are satisfied, there will be a solution. The grandparents of Irene are Lea and Dan.

Future Work

ODROID-XU4 shows the possibility for you to have your own intelligent server. In the next article, we are going to extend the ability of this intelligent system so that it can work with a sensor.

References

http://www.swi-prolog.org/

Ivan Bratko. Prolog programming for artificial intelligence, 2nd ed. Addison Wesley, J. Cassell, S. Prevost, J. Sullivan, and E. Churchill.

ODROID-VU

BACKLIGHT BRIGHTNESS CONTROL ON ANDROID

by Jörg Wolff

This guide shows you how to build the backlight Hardware Abstraction Layer (HAL) driver and the needed modifications in order to control the backlight of different displays. The driver is actually included in the latest ODROID-C2 Android image. This guide will show you how to modify the driver to build it for another board. Before following this guide, please ensure that you're using the latest Android NDK and the Android source tree.

Make a project directory wherever you like, but the preferred location would be inside NDK/samples. Make a subdirectory called "jni" and copy the "lights.c" code shown below into it.

Modify the "LOCAL_C_INCLUDES" in Android.mk to be the correct path of the Android source tree, then modify the "LOCAL_MODULE" in Android.mk to your board's name. Open a terminal window and navigate to the "jni" folder. Execute the command "path/to/ndk-build", typically in the directory where the NDK is installed.

You will find the library "liblights.odroidc.so" in folder "lib/armeabi/". Using adb, you can copy the library to the ODROID's folder "/system/lib/hw/":

```
adb remount
adb push ../libs/armeabi/liblights.odroidc.so /system/lib/hw/
adb reboot
```

Android will recognize the driver while booting. To enable this, the "boot.ini" must be edited in the following section.

Backlight HAL driver source code "lights.c":

```
/* Copyright (C) 2011 The Android Open Source Project
 *
 * Original code licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this software except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *   http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 *
 * This implements a lights hardware library for the Android emulator.
 * the following code should be built as a shared library that will be
 * placed into /system/lib/hw/lights.goldfish.so
 *
 * It will be loaded by the code in hardware/libhardware/hardware.c
 * which is itself called from
 * ./frameworks/base/services/jni/com_android_server_HardwareService.cpp
 *
 * Modified by J.  Wolff
 * Support of backlight control on Odroid board via pwm on pin 33.
 */

#include <android/log.h>
#include <stdint.h>
#include <string.h>
#include <unistd.h>
#include <errno.h>
#include <fcntl.h>
#include <pthread.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <stdbool.h>
```

```
#include <hardware/lights.h>
#include <hardware/hardware.h>

#include <assert.h>
#include <stdio.h>
#include <sys/stat.h>
#include <sys/syscall.h>
#include <stdlib.h>

#define  LOG_TAG    "lights.odroidc"
/* Set to 1 to enable debug messages to the log */
#define DEBUG 0
#if DEBUG
#define  LOGD(...)  __android_log_print(ANDROID_LOG_DEBUG,LOG_TAG,__VA_ARGS__)
#else
# define LOGD(...) do{}while(0)
#endif
#define  LOGI(...)  __android_log_print(ANDROID_LOG_INFO,LOG_TAG,__VA_ARGS__)
#define  LOGW(...)  __android_log_print(ANDROID_LOG_WARN,LOG_TAG,__VA_ARGS__)
#define  LOGE(...)  __android_log_print(ANDROID_LOG_ERROR,LOG_TAG,__VA_ARGS__)

#define BACKLIGHT "/sys/devices/platform/pwm-ctrl/duty0"
#define BACKLIGHT_EN "/sys/devices/platform/pwm-ctrl/enable0"
#define BACKLIGHT_FREQ "/sys/devices/platform/pwm-ctrl/freq0"

#define init_module(mod, len, opts) syscall(__NR_init_module, mod, len, opts)
#define delete_module(name, flags) syscall(__NR_delete_module, name, flags)

#define BACKLIGHT_PWM "backlight_pwm"
#define BACKLIGHT_PWM_YES "yes"
#define BACKLIGHT_PWM_NO "no"
#define BACKLIGHT_PWM_INV "invert"

static pthread_once_t g_init = PTHREAD_ONCE_INIT;
static pthread_mutex_t g_lock = PTHREAD_MUTEX_INITIALIZER;

char * env_backlight;
bool invert, enable;

void init_globals(void)
{
    LOGD( "in: %s", __FUNCTION__ );

    // init the mutex
    pthread_mutex_init(&g_lock, NULL);

    if (enable) {

        //pwm-meson
        int fd = open("/system/lib/modules/pwm-meson.ko", O_RDONLY);
        struct stat st;
        fstat(fd, &st);
        size_t image_size = st.st_size;
        void *image = malloc(image_size);
        read(fd, image, image_size);
        close(fd);
        if (init_module(image, image_size, "") != 0) {
            LOGE( "error loading pwm-meson.ko");
            return;
        }
        free(image);

        //pwm-ctrl
        fd = open("/system/lib/modules/pwm-ctrl.ko", O_RDONLY);
        fstat(fd, &st);
        image_size = st.st_size;
        image = malloc(image_size);
        read(fd, image, image_size);
        close(fd);
        if (init_module(image, image_size, "") != 0) {
            LOGE( "error loading pwm-ctrl.ko");
            return;
        }
        free(image);

        char value[20];
        int nwr, ret;
        fd = open(BACKLIGHT_FREQ, O_RDWR);
        if (fd > 0) {
            nwr = sprintf(value, "%d\n", 1000);
            ret = write(fd, value, nwr);
            close(fd);
        }

        fd = open(BACKLIGHT_EN, O_RDWR);
        if (fd > 0) {
            nwr = sprintf(value, "%d\n", 1);
            ret = write(fd, value, nwr);
            close(fd);
        }
    }
    LOGD( "leaving  %s", __FUNCTION__ );
}

static int
is_lit(struct light_state_t const* state)
{
    return state->color & 0x00ffffff;
}

/* set backlight brightness by LIGHTS_SERVICE_NAME service.  */
static int
set_light_backlight( struct light_device_t* dev, struct light_state_t const* state )
```

```
{
    int nwr, ret = -1, fd;
    char value[20];
    int light_level;

    if (!enable) {
      LOGD( "%s: Not implement-
ed.", __FUNCTION__ );
      return 0;
    }

    pthread_mutex_lock(&g_lock);
    light_level = state-
>color&0xff;
    light_level = light_level <<
2;
    if (light_level > 0) light_
level += 3;
    if (invert) light_level =
1023 - light_level;

    LOGD( "level: %d", light_lev-
el);

    fd = open(BACKLIGHT, O_RDWR);
    if (fd > 0) {
        nwr = sprintf(value,
"%d\n", light_level);
        ret = write(fd, value,
nwr);
        close(fd);
    }
    pthread_mutex_unlock(&g_
lock);
    return ret;
}

static int
set_light_buttons( struct light_
device_t* dev, struct light_
state_t const* state )
{
    /* @Waiting for later imple-
mentation.  */
    LOGD( "%s: Not implemented.",
__FUNCTION__ );

    return 0;
}

static int
set_light_battery( struct light_
device_t* dev, struct light_
state_t const* state )
{
    /* @Waiting for later imple-
mentation.  */
    LOGD( "%s: Not implemented.",
__FUNCTION__ );

    return 0;
}

static int
set_light_keyboard( struct light_
device_t* dev, struct light_
state_t const* state )
{
    /* @Waiting for later imple-
mentation.  */
    LOGD( "%s: Not implemented.",
__FUNCTION__ );

    return 0;
}
static int
set_light_notifications( struct
light_device_t* dev, struct
light_state_t const* state )
{
    /* @Waiting for later imple-
mentation.  */
    LOGD( "%s: Not implemented.",
__FUNCTION__ );

    return 0;
}

static int
set_light_attention( struct
light_device_t* dev, struct
light_state_t const* state )
{
    /* @Waiting for later imple-
mentation.  */
    LOGD( "%s: Not implemented.",
__FUNCTION__ );

    return 0;
}

/** Close the lights device */
static int
close_lights( struct light_
device_t *dev )
{
    free( dev );

    if (delete_module("pwm-me-
son", O_NONBLOCK) != 0) {
        LOGE("delete_module pwm-
meson");
        return EXIT_FAILURE;
    }

    if (delete_module("pwm-ctrl",
O_NONBLOCK) != 0) {
        LOGE("delete_module pwm-
ctrl");
        return EXIT_FAILURE;
    }

    return 0;
}
/**
 * module methods
 */
/** Open a new instance of a
lights device using name */
static int
open_lights( const struct hw_
module_t* module, char const
*name,
        struct hw_device_t **de-
vice )
{
    void* set_light;


    if (0 == strcmp( LIGHT_ID_
BACKLIGHT, name )) {
        set_light = set_light_
backlight;

          //check the bootargs
for backlight_pwm
          FILE * fp;
          char * line = NULL;
          char * list;
          char * value;
          size_t len = 0;
```

```
          fp = fopen("/proc/cmd-
line", "r");
          if (fp != NULL) {
                  getline(&line,
&len, fp);
                  list = strtok
(line, " ");
                  while (list !=
NULL)
                  {
                      LOGD
("%s\n",list);
                      if (0 ==
strncmp(BACKLIGHT_PWM, list, 13))
{
                          env_
backlight = strtok(list, "=");
                          env_
backlight = strtok(NULL, "=");
                          break;
                      }
                      list =
strtok (NULL, " ");
                  }
          }
        enable = false;
        invert = false;
          if (env_backlight !=
NULL) {
              LOGD("backlight_pwm
: %s", env_backlight);

              if (0 == strncmp(
BACKLIGHT_PWM_YES, env_backlight,
3 )) {
                      enable =
true;
              } else if (0 ==
strncmp( BACKLIGHT_PWM_NO, env_
backlight, 2 )) {
                      //enable =
false;
                  //return -EIN-
VAL;
              } else if (0 ==
strncmp( BACKLIGHT_PWM_INV, env_
backlight, 6 )) {
                      enable =
true;
                  invert = true;
              } else { enable =
false; }
          }
    } else if (0 == strcmp(
LIGHT_ID_KEYBOARD, name )) {
        set_light = set_light_
keyboard;
    } else if (0 == strcmp(
LIGHT_ID_BUTTONS, name )) {
        set_light = set_light_
buttons;
    } else if (0 == strcmp(
LIGHT_ID_BATTERY, name )) {
        set_light = set_light_
battery;
    } else if (0 == strcmp(
LIGHT_ID_NOTIFICATIONS, name )) {
        set_light = set_light_no-
tifications;
    } else if (0 == strcmp(
LIGHT_ID_ATTENTION, name )) {
        set_light = set_light_at-
tention;
    } else {
        LOGD( "%s: %s light isn't
supported yet.", __FUNCTION__,
name );
        return -EINVAL;
    }

    pthread_once(&g_init, init_
globals);

    struct light_device_t *dev
= malloc( sizeof(struct light_
device_t) );
    if (dev == NULL) {
        return -EINVAL;
    }
    memset( dev, 0, sizeof(*dev)
);

    dev->common.tag = HARDWARE_
DEVICE_TAG;
    dev->common.version = 0;
    dev->common.module = (struct
hw_module_t*)module;
    dev->common.close = (int (*)
(struct hw_device_t*))close_
lights;
    dev->set_light = set_light;

    *device = (struct hw_
device_t*)dev;
    return 0;
}

static struct hw_module_methods_t
lights_module_methods = {
    .open =  open_lights,
};

/*
 * The emulator lights Module
 */
struct hw_module_t HAL_MODULE_
INFO_SYM = {
    .tag = HARDWARE_MODULE_TAG,
    .version_major = 1,
    .version_minor = 0,
    .id = LIGHTS_HARDWARE_MOD-
ULE_ID,
    .name = "Odroid lights Mod-
ule",
    .author = "Amlogic",
    .methods = &lights_module_
methods,
};
```

Android.mk:

```
# Copyright (C) 2011 The Android
Open Source Project.
#
# Original code licensed under
the Apache License, Version 2.0
(the "License");
# you may not use this software
except in compliance with the
License.
# You may obtain a copy of the
License at
#
#   http://www.apache.org/licens-
es/LICENSE-2.0
#
# Unless required by applicable
law or agreed to in writing,
software
# distributed under the License
is distributed on an "AS IS" BA-
SIS,
# WITHOUT WARRANTIES OR CONDI-
```

```
TIONS OF ANY KIND, either express
or implied.
# See the License for the specific
language governing permissions
and
# limitations under the License.

LOCAL_PATH := $(call my-dir)

# HAL module implemenation, not
prelinked and stored in
# hw/<LIGHTS_HARDWARE_MODULE_
ID>.<ro.hardware>.so
include $(CLEAR_VARS)
LOCAL_C_INCLUDES += /path/to/an-
droid/source/tree/core/include
LOCAL_C_INCLUDES += /path/to/
android/source/tree/hardware/lib-
hardware/include
LOCAL_SRC_FILES := lights.c
LOCAL_PRELINK_MODULE := false
LOCAL_LDLIBS    := -llog
LOCAL_SHARED_LIBRARIES := libcu-
tils
LOCAL_MODULE := lights.odroidc
LOCAL_MODULE_TAGS := optional
include $(BUILD_SHARED_LIBRARY)
```

Application.mk:

```
NDK_TOOLCHAIN_VERSION=clang
APP_ABI := armeabi
APP_PLATFORM := android-24
```

The code can be found on Github at `http://bit.ly/2tshhFx`. Note that the frequency of the pwm is coded to 1KHz. You can modify the frequency in this code snippet:

```
fd = open(BACKLIGHT_FREQ, O_
RDWR);
if (fd > 0) {
    nwr = sprintf(value, "%d\n",
1000);
    ret = write(fd, value, nwr);
    close(fd);
}
```

It is also possible to overwrite the frequency at runtime:

```
$ echo 100 > /sys/devices/pwm-
ctrl.43/freq0
```

This can be useful if you like to experiment with the ADJ pin of a CCFL inverter, which is used with old LCD screens.

VU7+

To use the driver with your LCD screen, here are some examples. First, to connect the pwm signal from pin 33 of ODROID, a little tinkering is needed on the backside of the VU7+. The best practice is to use a wired resistor of about 330 Ohms, which should be glued to the board with superglue and soldered to pin 4 of the PT4103, or alternatively to the 10k smd resistor, marked with 103.

Figure 1 - Zoomed in view of VU7 changes

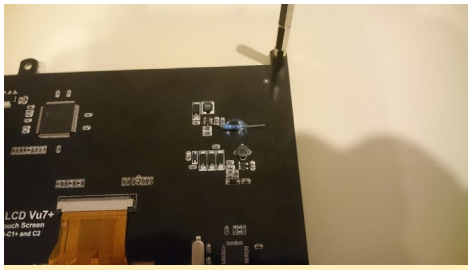


Figure 2 - VU7+ connection to CI

Note that it would work also without a resistor by using a direct connection. However, be careful and not burn our beloved hardware due to a human error, so we use a resistor.

For the VU7+, the following settings must be used in "boot.ini":

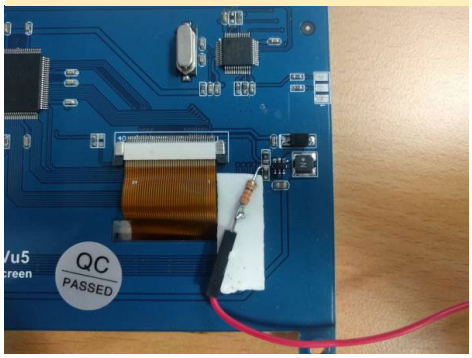
```
setenv hdmimode "1024X600p60hz"
setenv vout_mode "dvi"
setenv backlight_pwm "yes"
setenv bootargs "root=/
dev/mmcblk0p2 rw
console=ttyS0,115200n8 no_con-
sole_suspend vdaccfg=${vdac_con-
fig} logo=osd1,loaded,${fb_
addr},${outputmode},full
hdmimode=${hdmimode}
cvbsmode=${cvbsmode}
hdmitx=${cecconfig} vout=${vout_
mode} disablehpd=${disablehpd}
${disableuhs} androidboot.
serialno=${fbt_id#} ir_
remote=${ir_remote} usbcore.
autosuspend=-1 ${selinuxopt} sus-
pend_hdmiphy=${suspend_hdmiphy}}
backlight_pwm=${backlight_pwm}"
```

VU5 & VU7

The tinkering for these are identical to the VU7+. For the VU5 and VU7, we must change a few settings in the "boot.ini" file:

```
setenv hdmimode "800x480p60hz"
setenv vout_mode "dvi"
setenv backlight_pwm "yes"
setenv bootargs "root=/
dev/mmcblk0p2 rw
console=ttyS0,115200n8 no_con-
sole_suspend vdaccfg=${vdac_con-
fig} logo=osd1,loaded,${fb_
addr},${outputmode},full
hdmimode=${hdmimode}
cvbsmode=${cvbsmode}
hdmitx=${cecconfig} vout=${vout_
mode} disablehpd=${disablehpd}
${disableuhs} androidboot.
serialno=${fbt_id#} ir_
remote=${ir_remote} usbcore.
autosuspend=-1 ${selinuxopt} sus-
pend_hdmiphy=${suspend_hdmiphy}}
backlight_pwm=${backlight_pwm}"
```

Figure 3 - VU5 Modifications



VU8

The VU8 already has a connector for the pwm signal. For the VU8, the following settings must be used in "boot.ini":

```
setenv hdmimode "1024X768p60hz"
setenv vout_mode "dvi"
setenv backlight_pwm "invert"
setenv bootargs "root=/
dev/mmcblk0p2 rw
console=ttyS0,115200n8 no_con-
sole_suspend vdaccfg=${vdac_con-
fig} logo=osd1,loaded,${fb_
addr},${outputmode},full
hdmimode=${hdmimode}
cvbsmode=${cvbsmode}
hdmitx=${cecconfig} vout=${vout_
mode} disablehpd=${disablehpd}
${disableuhs} androidboot.
serialno=${fbt_id#} ir_
remote=${ir_remote} usbcore.
autosuspend=-1 ${selinuxopt} sus-
pend_hdmiphy=${suspend_hdmiphy}}
backlight_pwm=${backlight_pwm}"
```

Note that here "invert" on the VU8 is inverted, which means 100% duty translates to the backlight being off.

Controlling brightness with Java

To control the brightness from Java code, here are some code snippets. Be sure to add the necessary permission in AndroidManifest.xml file first. First, ask for permission <WRITE_SETTINGS> on Android Marshmallow:

```
<uses-permission
android:name="android.permission.
WRITE_SETTINGS" />
```

In code we can control it with the following snippet:

```
if (Build.VERSION.SDK_INT >=
Build.VERSION_CODES.M) {
    if (!Settings.System.canWrite
(getApplicationContext())) {
        Intent intent = new
Intent(android.provider.Settings.
ACTION_MANAGE_WRITE_SETTINGS);
        intent.setData(Uri.
parse("package:" + getPackage-
Name()));
        intent.addFlags(Intent.
FLAG_ACTIVITY_NEW_TASK);
        startActivity(intent);
    }
}
```

The following snippet reads the actual brightness value:

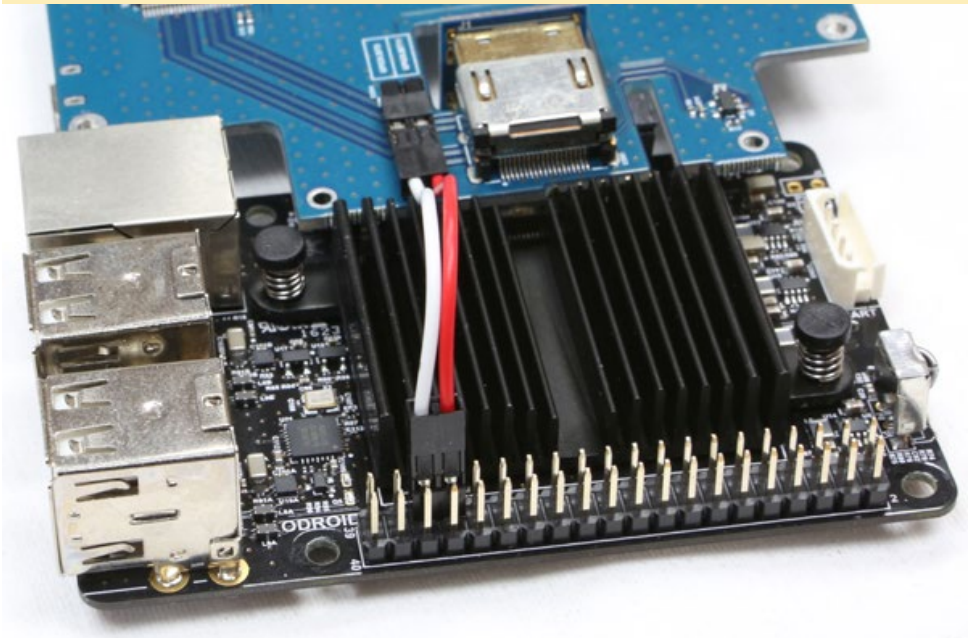
```
//Get the current system bright-
ness
try
{
    Settings.System.SCREEN_
BRIGHTNESS_MODE_MANUAL);
    int savedBright-
ness = Settings.System.
getInt(getContentResolver(),
        Settings.System.SCREEN_
BRIGHTNESS);
}
catch (Settings.SettingNotFound-
Exception e)
{
    Log.e("Error", "Cannot access
system brightness");
    e.printStackTrace();
}
```

Next, write the new brightness value:

```
int brightness = 50 // (0 … 100)
Settings.System.
putInt(getContentResolver(), Set-
tings.System.SCREEN_BRIGHTNESS,
brightness);
```

For more information please, please visit the forum thread at `http://bit.ly/2uSpczG`, and the Wiki page at `http://bit.ly/2vTKSbG`.

Figure 4 - VU8 connection



LINUX GAMING

FANBOY PART I – AM I A NINTENDO FANBOY?

by Tobias Schaaf

When it comes to retro gaming enthusiasts, there are generally two factions: Nintendo fanboys and SEGA fanboys. Back in the 80s and 90s, these were the two major players in the market. There were, however, a few other companies who also developed a fan base around their products, such as Atari, Commodore, and NEC. I'd like to take a personal look into these fan bases to see if I can identify myself with one or more of them in order to see what kind of fanboy I really am.

Nintendo Hardware

Nintendo is probably the biggest player from the past. If you think about it, they're probably the last console manufacturer from the 1980s that is still around and operating as a console manufacturer. There's no denying that Nintendo is a well-known company who made, and continues to make, good products. However, if someone was to ask if I was a Nintendo fanboy, I'd probably say "Nope, I never was." That's probably true, but why? Is it actually true?

Growing up in East Germany in the early 1980s, we did not have all the technical advantages of West Germany or other countries. Consoles were somewhat rare, and when they started becoming available in 1989 and into the early 1990's, only a few people had access to them. There was no hype around getting whatever console came first, since both the Nintendo Entertainment System (NES) and the SEGA Master System were already well established during that time. People just picked whichever one they wanted. While I didn't know too many people who owned consoles at that time, one of my uncles did indeed have an NES, which I was lucky enough to play every now and then.

So which games did I like to play? Super Mario Bros was okay, but I never got deeply into it. Did I enjoy it? More than likely, since I was a child, but it definitely was not my favorite game.

What was my favorite, then? Nintendo World Cup was a lot of fun for me as a child, and still is today. As a soccer game it had very few rules. You could tackle your enemies, make them fall down, do trick shots etc. It was fun, the controls were easy, and you soon figured out how to beat the enemy. It was a lot of fun just knocking out all of the opposing players and watching them try to pass balls to players laying on the ground. It also had an awesome soundtrack, which is probably one of the best in the entire NES library. Like I said, I never played a lot of NES games and may have missed some good ones as a child. But there are good reasons for my saying that I'm not a Nintendo fanboy.

I recently discovered that one of my first consoles, or, more accurately, handheld games, was actually from Nintendo as well, although only a few would recognize it as an NES game. "Egg" was part of Nintendo's Game & Watch series. Game & Watch was a series of handhelds that operated as both alarm clock and gaming device, although I highly doubt anyone really ever used it as a clock. This game was very simple: just four directional buttons that players used to catch eggs as they fell.

"Egg", a Game & Watch game from Nintendo

As we all know, Nintendo created a lot of consoles and handhelds, such as the Game Boy. Although I knew some people at school who had a Game Boy, I never really found it interesting. It had no colors, no backlight, and the graphics were not very sharp. I saw one guy who had a Sega Game Gear. It had colors, and you could play your SEGA Master System games on it, and it had a TV receiver. I'm sorry, what was that Game Boy thing you were talking about?

I didn't know anybody with a "good" Nintendo device back then. I've never known anyone with a Super Nintendo, and I've never seen a Game Boy Color. I never even heard of the Game Boy Advance until much later. When it came to Nintendo, I missed everything between the NES/Game Boy and the Nintendo DS. That's probably a big reason why I don't consider myself a Nintendo fanboy,

with the exception of the Nintendo 64, which had a demonstration console in our local shopping mall. As a child, I'd play N64 games while my parents shopped. It was fun, even if you had to share with children you didn't know, but I never actually owned the console and there were not a lot of games on display except for Super Mario 64.

As an adult, I got an Nintendo DSi XL, and it is a very fun handheld system. I also got a Nintendo Wii, and while that was great for casual gaming with friends or alone, even the sports games, the graphics were really not all that great. I actually enjoyed, and still greatly enjoy, the Nintendo DS. There are a lot of fun games including RPGs, adventure, and strategy games which I really like, and the graphics are good enough. It's a very fun handheld.

I also enjoyed games like Time Hollow, the Ace Attorney series, Another Code:Two Memories, Advance Wars, the Cooking Mama series, the Final Fantasy series, the Luminous Arc series (I love this one), several Dragon Ball Z games, Bleach the 3rd Phantom (one of my favorite games), Summon Night: Twin Age, Suikoden Tierkreis, the Shin Megami Tensei series, Infinite Space, and so much more. There are so many awesome games on the Nintendo DS that it's really hard to pick out just a few favorites.

The Wii had some nice games I enjoyed playing; mostly the Rayman Raving Rabbids series, which is amazing as a party game, but also some more serious games, such as de Blob 1 and 2, Resident Evil 4, Mario Kart Wii (which is much better than the N64 version), MadWorld, Xenoblade Chronicles, Red Steel 2, Overlord, No More Heroes, The Last Story, Pandora's Tower, and many more.

Nintendo emulation on ODROID

When I started working with emulation on the ODROID back in 2012, the possibilities were rather limited. There was no 3D support, Lakka did not exist, and RetroArch was still mostly unheard of at that time.

There were a couple of standalone emulators based on Simple DirectMedia Layer (SDL) that more or less worked, but the big breakthrough came with Mednafen. Mednafen is a multi-system emulator that allowed you to emulate all kinds of consoles and handhelds at amazing speeds. Still, it had its downsides. Super Nintendo wouldn't run at a decent speed and neither would PlayStation games, but many other platforms worked fine.

While I still wasn't into NES games, I found my passion for the Game Boy Advance playing games like Riviera: The Promised Land, different Dragon Ball Z games, Advance Wars, Medabots, Summon Night 1 and 2, and more. Even later, when the 3D GPU driver was available and Super Nintendo emulation was finally working, I was not particularly interested in Super Nintendo games. Up to that point, I only played a few games for the Super Nintendo. I personally like the Game Boy Advance better than the Super Nintendo. Aside from the resolution, the Game Boy Advance is much better than the Super Nintendo in any field.

Many games from the Super Nintendo also got released for the Game Boy Advance, often with improved graphics and sounds. Therefore, I usually played Game Boy Advance versions of a game, since they often were better than the Super Nintendo version, in my opinion.

I really enjoyed the Game Boy Advance, and some of my favorite games of all time are either Game Boy Advance games, or games I played for the first time on the Game Boy Advance. There are a few Super Nintendo games that I like a lot, such as E.V.O.: Search for Eden, but when it comes to Nintendo, I usually end up choosing games for the Game Boy Advance.

There are also other Nintendo consoles and handhelds available for emulation on ODROID, including Virtual Boy, Nintendo 64, Nintendo DS, and more. These emulators work quite well, but none of them get near the amount of play time as the time I play the Game Boy Advance emulators.

There are a few interesting games for the Nintendo 64, but overall, I'm not a fan. I really enjoy the Nintendo DS, but using a mouse or gamepad instead of a touch input means that the experience using the NDS emulator is quite different from the "real thing". However, there are some games on the Nintendo DS that are worth playing on ODROID, and I would say it's probably my second-favorite emulator for the ODROID. The emulation is not perfect, since it has some speed issues, which hopefully can be solved at some point in the future, but it is still fun, especially on the ODROID-XU3/XU4.

Final thoughts

Although I've had some good experiences with the Game Boy Advance and Nintendo DS and, to a lesser extent, the N64 and the Wii, I still don't consider myself a Nintendo fanboy. This could be due to having little exposure to Nintendo as a child, but I feel also like I've just had better, or at least different, options available.

This doesn't mean that Nintendo doesn't deserve credit. On the contrary, the Game Boy Advance and Nintendo DS are really great machines. Current consoles like the Wii U and Switch will likely have a positive impact on the gaming industry. Unfortunately, I'm not familiar with the GameCube, but I feel I could could get some enjoyment out of it as well.

Still, options for Nintendo emulation on ODROID remain limited to Game & Watch, Game Boy and Game Boy Color, NES, Virtual Boy, Super Nintendo, Game Boy Advance, Nintendo 64, and Nintendo DS. That is quite a number of consoles, but not many that I'm eager to play. In conclusion, I stand with my original premise: I like Nintendo, but I'm not a fanboy.

MIGRATING FROM UBUNTU MATE TO LUBUNTU

A STEP-BY-STEP GUIDE FOR SWITCHING TO AN LXDE DESKTOP

by Miltiadis Melissas

Hardkernel provides excellent releases for every board it produces. The latest image for the ODROID-XU4 (`http://bit.ly/2utQxEE`) is Ubuntu 16.04 with the Mate desktop as a graphical user interface (GUI). For the last couple of years, the company has preferred to provide public releases of Linux/Ubuntu to users with the Mate desktop. This is no accident, since Mate is a great GUI environment thanks to the hard work work done by its development team. Nevertheless, as good as it gets, occasionally users need a change of pace. This simple guide provides exactly that: a brief, step-by-step description of how to switch from the Mate desktop to its LXDE equivalent, transforming Ubuntu to a fully-functional Lubuntu variant. Let's take a look at how to make this happen.

Installation

First, download the Software Release for Linux/Ubuntu (v2.0) Kernel 4.9 XU3/XU4 from Hardkernel's excellent XU4 wiki page (`http://bit.ly/2utQxEE`) and write it to an eMMC or microSD card. The main features of this release include:

- Linux Kernel LTS 4.9.27
- Ubuntu 16.04.2
- Updated Mali GPU driver to the latest version (r17p0)

After the first boot, you will receive a message that reads "The RootFS Auto-resize feature has changed!!! Once everything is done after auto-reboot, the power will turn off. Wait a couple of minutes. Please press the power button if the blue LED is off."

As soon as this image is booted up, login in with the following access credentials, as shown in Figure 1:

```
$ user/root: odroid
$ password: odroid
```

Wait for a few seconds for the system to load your newly made Mate desktop environment, as shown in Figure 2.

Figure 1 - Login screen

Figure 2 - Mate desktop environment

First things first: it's a good idea to update/upgrade your system. It's also very easy. Open a terminal window (Ctrl+Alt+T) and type the following Linux commands, one by one, allowing the system time to finish each step before moving on to the next:

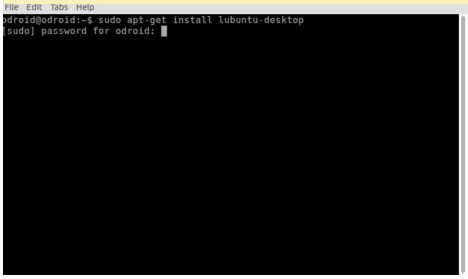
```
$ sudo apt-get update
$ sudo apt-get upgrade
$ sudo apt-get dist-upgrade
```

Before rebooting, you will also need to update the Kernel software:

```
$ sudo apt-get install \
  linux-image-xu3
$ sudo reboot
```

After rebooting, login again and then open a terminal window (Ctrl+Alt+T). Install the Lubuntu desktop by typing the following command, as shown in Figure 3.

Figure 3 - Installing the Lubuntu desktop



```
$ sudo apt-get install lubuntu-desktop
```

At this point, it's a good idea to have a drink or a satisfying snack while the Lubuntu desktop is downloaded and installed onto your system. As soon as the installation is complete, reboot your computer.

```
$ sudo reboot
```

It is now time to customize your new Lubuntu desktop, as shown in Figure 4.

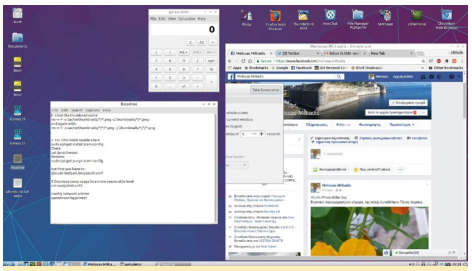


Figure 4 - Lubuntu desktop environment

Modify time and date

Access the Start menu by clicking on the LXDE icon, then select System Tools > Time and Date. In the new window, set the appropriate time and day, as shown in Figures 5 and 6.

Install languages

Installing the system's languages is also easy. From the Start menu, select Preferences > Language Support and install the appropriate languages for your system. English is installed as the default language.

Figure 5 - Opening the Lubuntu System Tools

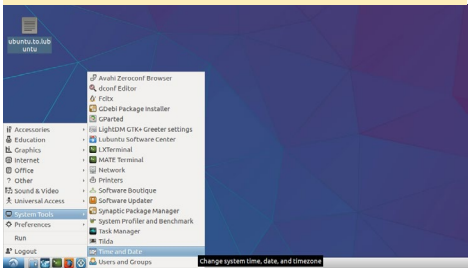
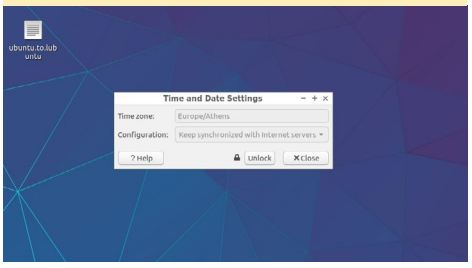


Figure 6 - Setting the time and day



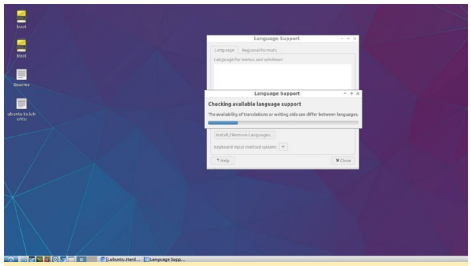


Figure 7 - English is the default language for Lubuntu

Add keyboard layouts

If you are going to use Lubuntu as a fully operational desktop, you will need to select a keyboard layout, especially if English is not your native language. The process is a bit different this time. Right-click on the panel and select Add/Remove Panel Items. From there, click Add and choose Keyboard Layout Handler, as shown in Figures 8 and 9.

Add panel items

This last step is mainly for cosmetic purposes. You can add a "Keyboard LEDs" indicator to make your life a bit easier by right-clicking the panel and again choosing Add/Remove Panel Items. From there, select Keyboard LEDs and check each box to enable all the LEDs, as shown in Figure 10

Figure 8 - Opening the Add/Remove Panel Items feature

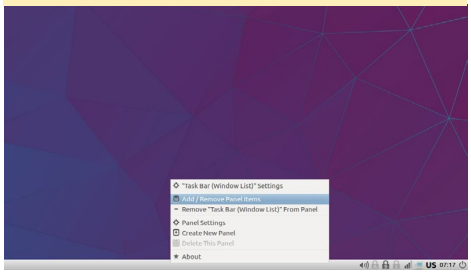
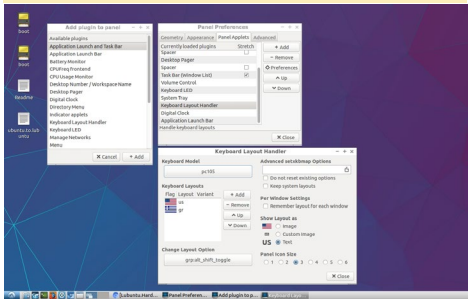


Figure 9 - Choosing the keyboard layout



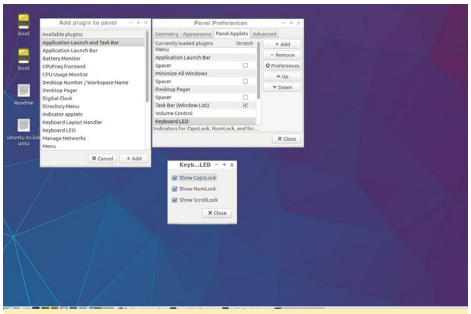


Figure 10 - Enabling the keyboard LEDs

Notes

I hope you enjoyed this simple step-by-step guide to transforming Ubuntu to Lubuntu, and now it's time for you to give it a try. I would love to receive your comments on my twitter account (@Miltos01). Changes in life are always necessary and important to keep it interesting, and the same goes for Ubuntu!

USING THE ADAFRUIT I28X64 OLED BONNET ON AN ODROID-CI+

PROGRAMMING WITH LUMA.OLED AND WIRINGPI

by Dennis Chang (@dchang0)

There are many small I2C or SPI OLED display boards on the market, but most of them are targeted at the Arduino market, and thus have pin ordering inconvenient for use on ODROIDs. At the time of this article, I have only seen three I2C OLED display boards on the market designed specifically to fit directly on the Raspberry Pi GPIO header pins. All three of them are monochrome and based on the SSD1306 OLED/PLED driver chip.

Since fellow ODROID forum member @eudoxos had a bad experience with an Adafruit 128x32 monochrome OLED display breakout board, I did not want to take a chance on the two of the three choices that are based on the same 128x32 display. A few forum members had confirmed that a generic I2C 128x64 OLED display worked, so I figured Adafruit's 128x64 OLED Bonnet/pHat (product ID 3531) would work. It does, and here is how.

Like most bonnets or pHats, the Adafruit 3531 has a low-profile pass-through SMT 5mm female 2x20 header on the bottom. This is about 2mm too short for the 3531 to fit securely if placed directly onto the GPIO pins on my ODROID C1+ because of the height of the heatsink. I had to use a stacking 2x20 header as an extension to raise the 3531 above the heatsink.

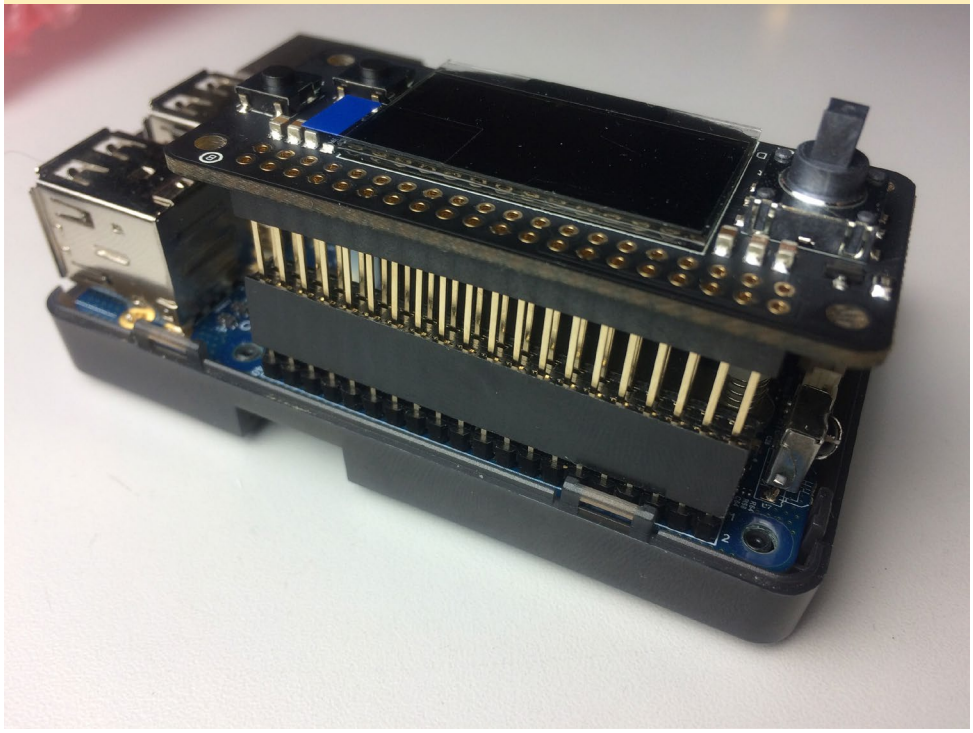
For testing, I did not bother pushing the headers fully together, as shown in Figure 1. If I had, the bonnet would sit just 1mm above level with the tops of the USB port shields, and the male pins of the stacking header would be pushed through the bonnet to be exposed and accessible on top. It would be possible to avoid using the stacking header if I desoldered the SMT low-profile header on the 3531 and put a standard-height or extended-height female header in its place.

I started with my trusty ODROID C1+, re-imaged with Ubuntu 16.04 minimal and running in headless mode. To get the Adafruit 3531 display working, I used the same driver library that others confirmed to be working, which is the Luma.OLED (`http://bit.ly/2tSA5gP`).

Note that I did not actually build from source. I read the installation instructions at `http://bit.ly/2gVUCj2` and figured out what works:

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

Figure I - Adafruit 3531 mounted on the ODROID-CI+



```
python-pip libfreetype6-dev
libjpeg-dev
$ pip install --upgrade pip
$ pip install mock
$ pip install pytest
$ pip install --upgrade luma.oled
```

The install went well, and mock and pytest are installed as prerequisites for Luma.OLD. Make sure to stay in the sudo -s session for the rest of the steps in this article.

Type the following command to enable I2C:

```
$ modprobe aml_i2c
$ echo "aml_i2c" | sudo tee /etc/
modules
```

You can optionally install and use i2cdetect to scan the I2C bus for the display:

```
$ apt-get install i2c-tools
$ i2cdetect -y 1
```

This is how the Adafruit 3531 shows up in the scan (as address 0x3c):

```
     0  1  2  3  4  5  6  7  8  9
a  b  c  d  e  f
00:          -- -- -- -- -- -- --
-- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- --
-- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- --
-- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- --
-- -- 3c -- -- --
40: -- -- -- -- -- -- -- -- -- --
-- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- --
-- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- --
-- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

I immediately tested the display with this simple Python code from the instructions at:

`http://bit.ly/2tsEQ10`. Save the following source code as testdisplay.py:

```
from luma.core.interface.serial
import i2c, spi
from luma.core.render import
canvas
from luma.oled.device import
ssd1306, ssd1325, ssd1331, sh1106

# rev.1 users set port=0
# substitute spi(device=0,
port=0) below if using that in-
terface
serial = i2c(port=1,
address=0x3C)

# substitute ssd1331(...) or
sh1106(...) below if using that
device
device = ssd1306(serial)

with canvas(device) as draw:
    draw.rectangle(device.
bounding_box, outline="white",
fill="black")
    draw.text((30, 30), "Hello
ODROID", fill="white")

raw_input()
```

To run the above program, type the following command:

```
$ python testdisplay.py
```

A big plus is that the Adafruit 3531 includes two buttons and a 5-way joystick, something I really needed for a battery-powered, portable C1+-based project that needed a simple menu interface. The ODROID 3.2 TFT touchscreen consumed too much power and was too hard to operate while walking about, but the buttons and joystick would have been perfect.

To get the joystick and buttons working, I used HardKernel's port of wiringPi to the ODROID. Install Git if it is not already available:

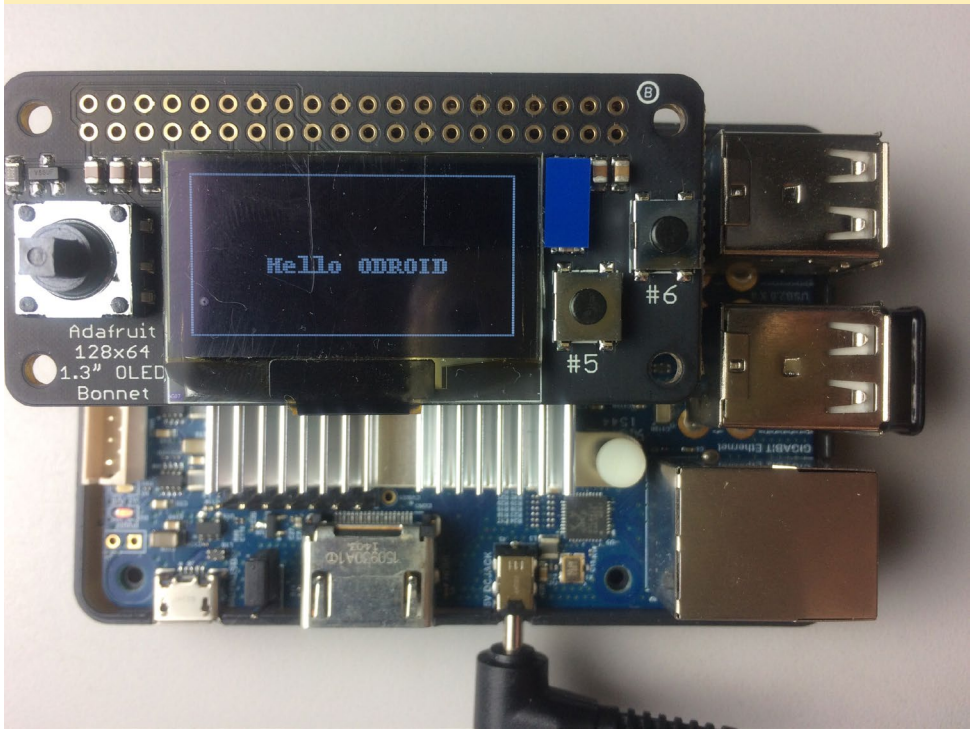
```
$ apt-get install git
```

Install swig3.0 and python-dev:

```
$ apt-get install python-dev
python-setuptools
$ apt-get install swig3.0
```

Get and build HardKernel's wiringPi 2 for Python:

Figure 2 - Demo WiringPi project running on the ODROID-C1+



ODROID-XU4 LED CONTROL

edited by Rob Roy

We previously featured an article on how to control the LEDs on the ODROID-U3 (http://bit.ly/2vpLLNj). You can have similar control on the XU4 blue LED using a trigger node in the /sys directory. The red LED is hard-wired to the power input rail and so is not able to be controlled via software.

1. Turn off the blue LED:

```
$ su -
$ echo none > /sys/class/leds/
blue\:heartbeat/trigger
```

2. Turn on the blue LED:

```
$ su -
$ echo default-on > /sys/
class/leds/blue\:heartbeat/
trigger
```

3. Show a heartbeat on the blue LED (factory setting):

```
$ su -
$ echo heartbeat > /sys/class/
leds/blue\:heartbeat/trigger
```

There are many other trigger modes, although not all of them are enabled. They can be listed using the following command:

```
$ cat /sys/class/leds/
blue\:heartbeat/trigger
```

To make the change take effect on every boot, add a line to /etc/rc.local. For example, to turn off the blue LED permanently, add the following snippet to /etc/rc.local:

```
echo none > /sys/class/leds/
blue\:heartbeat/trigger
```

```
$ git clone https://github.com/
hardkernel/WiringPi2-Python.git
$ cd WiringPi2-Python
$ git submodule init
$ git submodule update
$ swig3.0 -python -threads
wiringpi.i
$ python setup.py build install
```

Through trial and error, I discovered the correct pin numbers for the Adafruit 3531's buttons and joystick. They are very different from the pin numbers provided by Adafruit for use with wiringPi for the Raspberry Pi.

Here is the source code for the demonstration of the buttons, which should be saved as hello.py:

```
from luma.core.interface.serial
import i2c, spi
from luma.core.render import
canvas
from luma.oled.device import
ssd1306, ssd1325, ssd1331, sh1106
import wiringpi2 as wpi
import time

# rev.1 users set port=0
# substitute spi(device=0,
port=0) below if using that in-
terface
serial = i2c(port=1,
address=0x3C)

# substitute ssd1331(...) or
sh1106(...) below if using that
device
device = ssd1306(serial)

with canvas(device) as draw:
    draw.rectangle(device.
bounding_box, outline="white",
fill="black")
    draw.text((30, 30), "Hello
ODROID", fill="white")

INPUT = 0
PUD_UP = 2
L_pin = 2
R_pin = 4
C_pin = 7
U_pin = 0
D_pin = 3
A_pin = 21
B_pin = 22
buttons = [0, 2, 3, 4, 7, 21, 22]

wpi.wiringPiSetup()

for x in buttons:
      wpi.pinMode(x, INPUT)
      wpi.pullUpDnControl(x, PUD_
UP)
      wpi.digitalWrite(x, 0)

while True:
      time.sleep(0.05)

      for x in buttons:
            print "x: %d state:
%d" % (x, wpi.digitalRead(x))
```

Pressing and holding buttons will change the state from 1 to 0. The following output is with the joystick held right:

```
x: 0 state: 1
x: 2 state: 1
x: 3 state: 1
x: 4 state: 0
x: 7 state: 1
x: 21 state: 1
x: 22 state: 1
```

The above output verifies that the code works! The buttons and joystick pin numbers are listed in the source code above.

CRYPTOCURRENCY MINING

A VIABILITY PROJECT AND STABILITY TEST FOR KERNEL VERSION 4.9 ON AN ODROID-XU4 HIGH PERFORMANCE COMPUTING CLUSTER

edited by Rob Roy

Hardkernel used crypto currency mining to test the XU4 Kernel 4.9.27 stability earlier this year. Twenty XU4 boards, set up as a supercomputing cluster, have been running the Verium coin (VRM) mining software to utilize all 160 CPU cores and 40GB RAM as much as possible. After two weeks of intensive testing, Hardkernel can confidently say that the Kernel 4.9 LTS on XU4 is quite stable. Check out the demo video at https://youtu.be/wbffhvn_J4E.

We had to reboot all of them two times to update the Kernel when 4.9.28 and 4.9.30 were released. A couple of units had to be manually reset because of 5V DC cable issue, so we added some more power wires to reinforce the power rails. Other than that, there was no other issues.

The cost of setting up a cluster is detailed below, for a total of approximately USD $1500:

20 x XU4 = USD$1,180
20 x 8G microSD cards = USD$160
20 x LAN cable = USD$10
1 x 5V/80A PSU (`http://amzn.to/2vkKjus`) = USD$45
1 x 24 port network switch (`http://amzn.to/2vfKQyc`) = USD$41
Cables, plugs, and PCB spacers = $30
Optional AC power gauage meter = $30

We measured the power rate, and its accumulated power is 65KWh and monthly power consumption could be 130 KWh. The estimated monthly electricity cost will be around USD$15. We earned only 100 VRM in two weeks. If we consider some exchange fees for changing VRM to BTC to US currency, its actual value is $90 at this moment. Our estimated monthly income will be USD$90 x 2 - USD$15 = USD$165. We have to keep running this the test equipment for about 9 months to get an even return on investment (ROI).

We also learned how much faster the XU4 is than the Raspberry Pi 3. The average hash rate (Hashes per minute) on XU4 is 390H/min, while the RPi3 is only 110H/min, which means that the XU4's computing power is 3.5x higher than the RPi3. For further reading about the mining software setup and internal algorithms, please refer to the following articles: *CPU Mining is back! A complete how to guide and profit analysis for Verium mining on a farm of single board computers - Part 1*: `https://goo.gl/XM5ype`, *Part 2a*: `https://goo.gl/c38MWj`, *Part 2b*: `https://goo.gl/4Gi8ax`

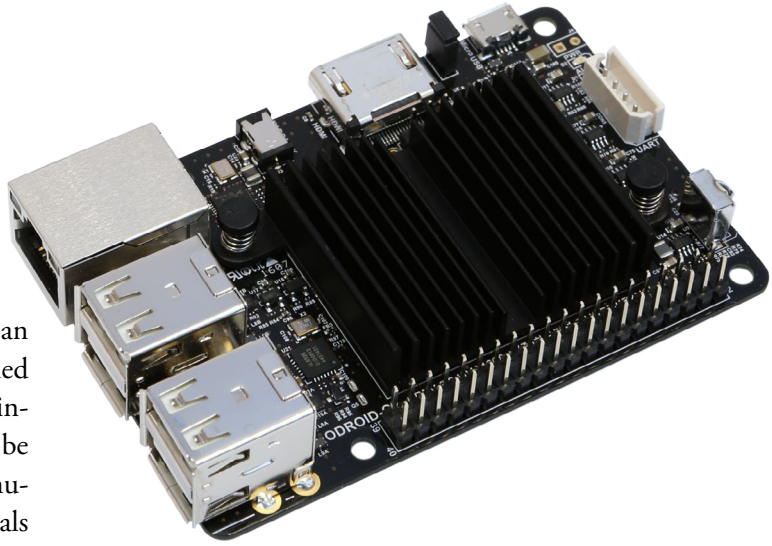
Wallet download: `http://www.vericoin.info`
Miner: `https://github.com/effectsToCause/veriumMiner`

For comments, questions, and suggestions, please visit the original thread at `http://bit.ly/2uh0ncy`.

GETTING STARTED WITH ANDROID ON THE ODROID-C2

A BEGINNER'S GUIDE

edited by Rob Roy

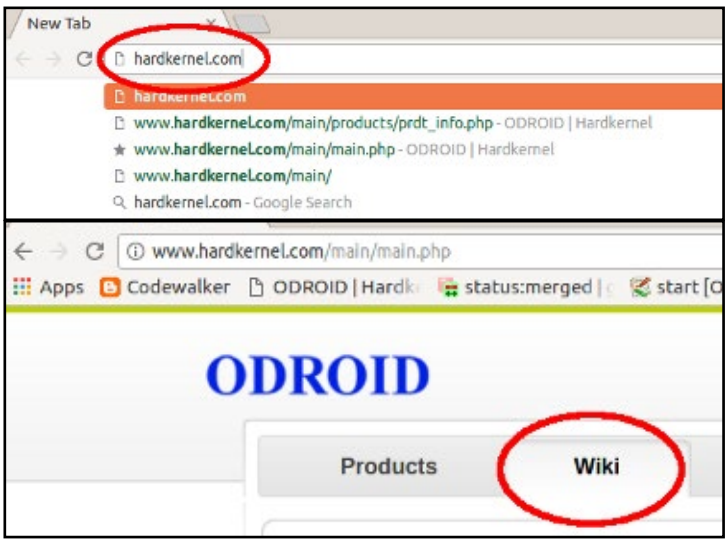


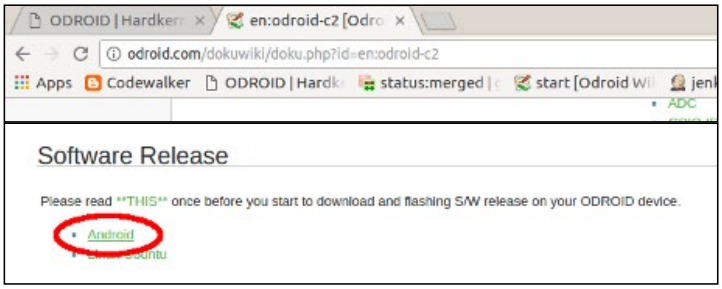
There are two options for installing Android on an ODROID-C2. Hardkernel offers a pre-installed eMMC or microSD card, which would only require installing Google Play. Alternatively, the Android OS may be downloaded from the Hardkernel website and installed manually onto the eMMC or microSD card. The required materials for running Android on an ODROID-C2 are listed below:

- **ODROID-C2** (`http://bit.ly/1oTJBya`)
- **5V/2A Power supply (US:** `http://bit.ly/2ugY0Xe`**, EU:** `http://bit.ly/1X0bgdt`**, Worldwide:** `http://bit.ly/OhMyWx`)
- **Memory card pre-installed with an operating system (eMMC:** `http://bit.ly/2vq2TCq`**, microSD card:** `http://bit.ly/2u1fM5I`)
- **HDMI cable:** `http://bit.ly/2uSu3Ay`
- **Monitor or TV with an HDMI port**

Watch the video `https://youtu.be/fEyeMTS3idU` at to see how easy it is to get started! If you do not have a memory card pre-installed with an operating system, please follow instructions below to install it onto the memory card.

In addition to all the items listed above, you will need a PC



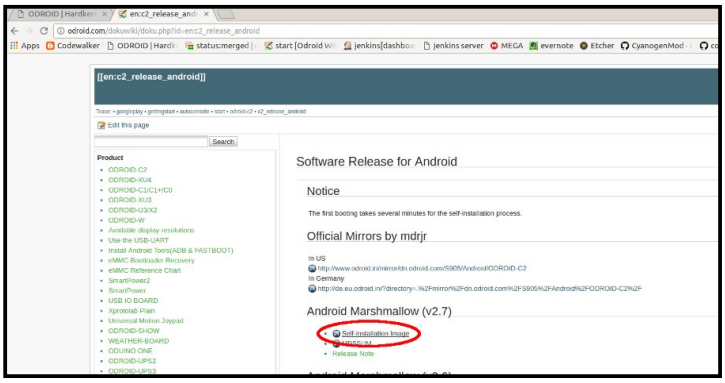


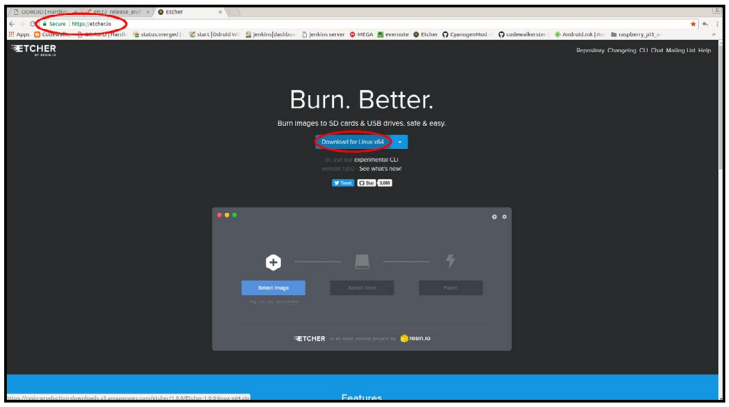
in order to install Android OS to the memory card. An instructional video is available at `https://youtu.be/9Zi2_OTSl_I` and `https://youtu.be/NyQif1j2WkA`. Note that the Smart Power 2 power supply (`http://bit.ly/2j3hhcv`) is used in the video.

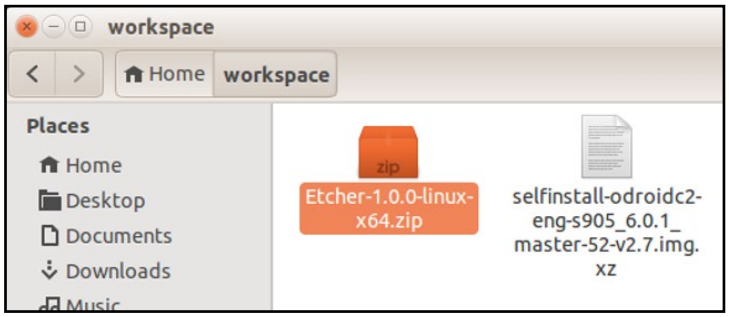
First, download the Android operating system from the Hardkernel website at `http://bit.ly/2vkGwgX`. Make sure to wait for the complete download. To install, or "flash", Android to the memory card, we recommend using Etcher, as described at `http://bit.ly/2f61k5x`. You can download etcher from `https://etcher.io/`. Etcher works on Mac OS, Linux and Windows, and is the easiest option for most users. Etcher also supports writing OS images directly from the zip file, without any unzipping required. To install the OS on an eMMC module, you will need an eMMC module reader (`http://bit.ly/2ugIKK8`) and a USB multi reader (`http://bit.ly/2vpTv1y`) to connect it to your PC.

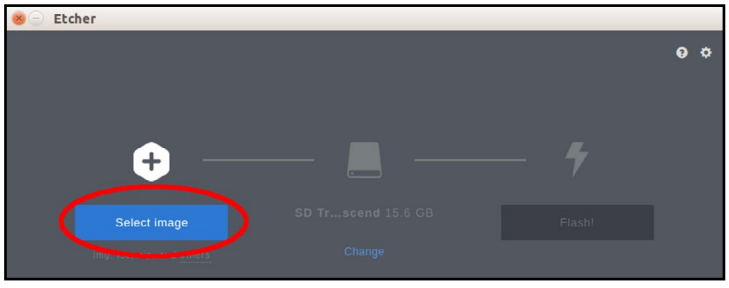
To install Android on an eMMC, follow the instructional video at `https://youtu.be/XfJY4KxLxps`. If using a microSD card, watch `https://youtu.be/SnrqyoUBry4`.

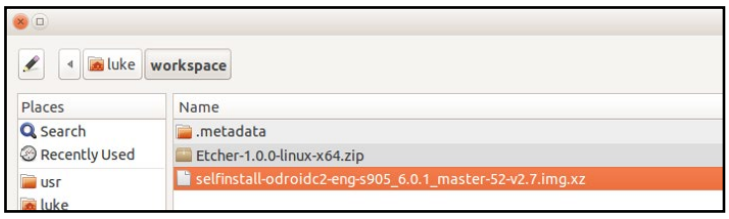
When OS installation is complete on the memory card, connect the HDMI cable to your ODROID-C2, then plug

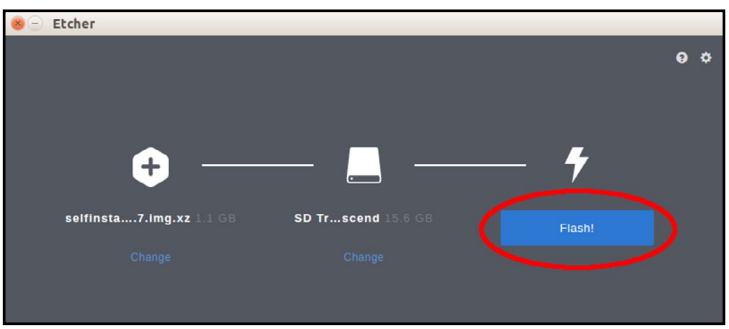












the power supply. After a few seconds, you will see the home screen of Android. For more information, please visit the original Wiki article at article at `http://bit.ly/2uhhlrj`.

Installing Google Play

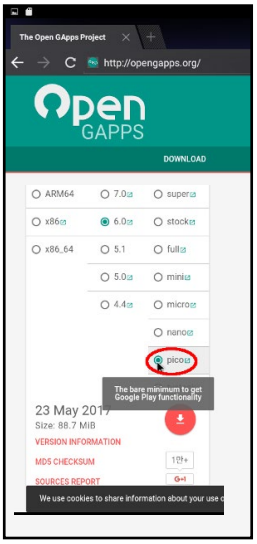
To install Google Play onto an ODROID-C2, the following items are required:

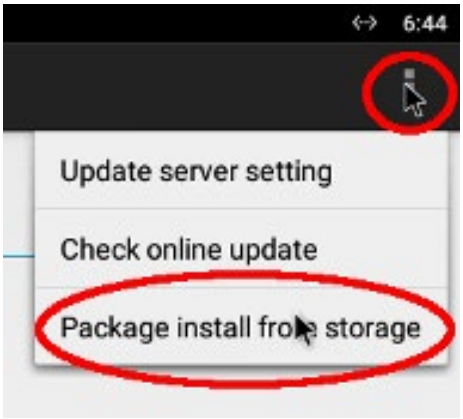
ODROID-C2 (`http://bit.ly/1oTJBya`)

Internet connected via Ethernet cable (`http://bit.ly/2vg6v9I`) or WiFi module (`http://bit.ly/22nyxra`)

If you want to download the Google Play to a PC and transfer it to the C2, you will need to connect the C2 to PC via an OTG cable (`http://bit.ly/2vqf6H5`).

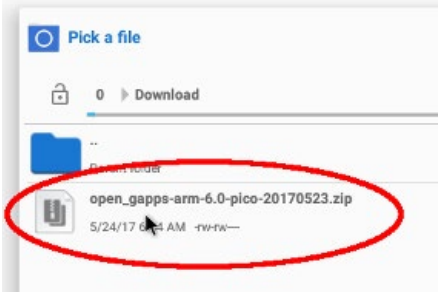
An instructional video is available at `https://youtu.be/PKO8ZKJM_0c`. The images below highlight the main steps in the video. Open the browser on ODROID-C2 and visit `http://opengapps.org`. We recommend using the "pico" version, but the ODROID-C2 also supports micro and nano versions.





The video at `https://youtu.be/wOhAgkkWnjI` shows how to login to your Google account and open Google Play.

For more information, please visit the original Wiki article at `http://bit.ly/2vqgz0c`.



MEET AN ODROIDIAN

MARTÍ BONAMUSA, REAL-TIME 3D DATA ANALYTICS ENTREPRENEUR

edited by Rob Roy (@robroy)

Please tell us a little about yourself.

I am 39 years old and was born in Sant Celoni, a town 50km north of Barcelona in Catalonia. I am married to my wife Fiona and have 2 children, Joan and Ferran, who are 8 and 6 years old. We all live in Santa Maria de Palautordera, which is a small village next to Sant Celoni. Fiona is a teacher at a school of students with different levels of disability.

Martí's Family: Fiona, Joan and Ferran

Since I was very young, I have had a great passion for electronics and programming, and once I finished my basic school, I decided to study electronics and earned a degree in electronics Engineering from the Universidad Politècnica de Catalunya in 2001. I later earned a senior electronics engineering degree from the Universitat Autònoma de Barcelona in 2004.

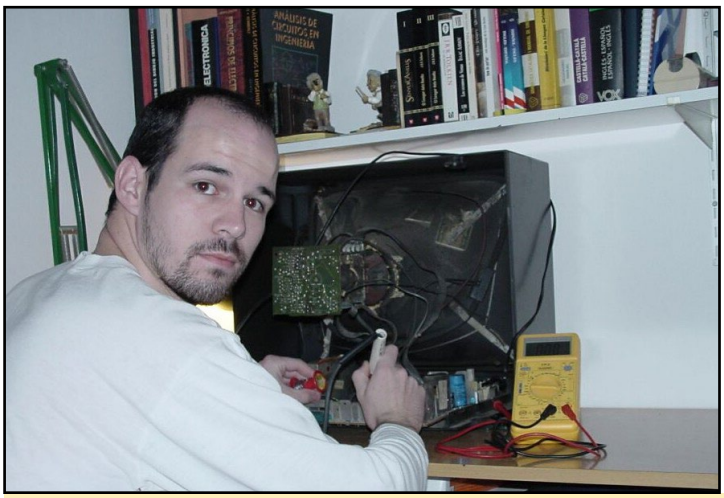
As a result of different experiences in the field of machine vision, I founded the company OnTrace (`www.on-trace.com`) with Josep Mesado, who was a colleague of mine at University. The reason for creating the company was that we detected a market need related to the analytics of people behavior in physical spaces. Taking into consideration high precision and operation in different environments as a main factors, we designed a 3D stereoscopic camera. That is when we started to work with ODROID computing devices as a component of our smart cameras.

How did you get started with computers?

My passion for electronics started at a very early age, and at 9 years old, I started making my first inventions with a game called "Scatron". When I was 12, my parents bought me a Spectrum ZX as a gift to play games, but it allowed me to start programming. By the time I was 15 years old, I had already done several programs with BASIC.

The Scatron game got Martí started with electronics

At the age of 16, when I started studying electronics, I discovered microcontrollers and started playing with them, especially with the PIC and 8031, and programming in assembler was fascinating for me.



Martí repairing a television in 2001

What attracted you to the ODROID platform?

I started looking for embedded platforms, and mainly focusing on low power consumption and high performance, and here is where I discovered the ODROID platform. We first tested the X2, and came to the conclusion that it fit our needs.

We evolved from the X2 to the XU4 which offered higher performance, and now with the new C2, it permits us to really improve our devices and guarantee our market needs. Great support and forums are also very useful and important to us.

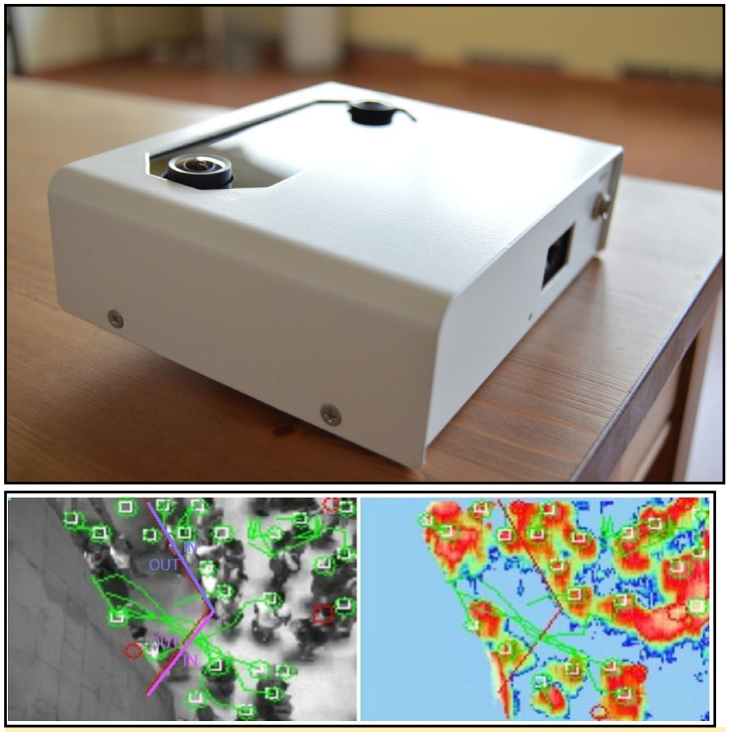
How do you use your ODROIDs?

Currently we have several different products, with three of them based on ODROIDs as a computing device:

A people counting device based on 3D camera and the XU4 (image: people counter3d.jpg)

A face recognition device using the ODROID-C1

A new version of 3D camera with PoE connectivity using the ODROID-C2



Top: People Counter 3D device)
Bottom: People Counter graph analyzing a busy area

Which ODROID is your favorite and why?

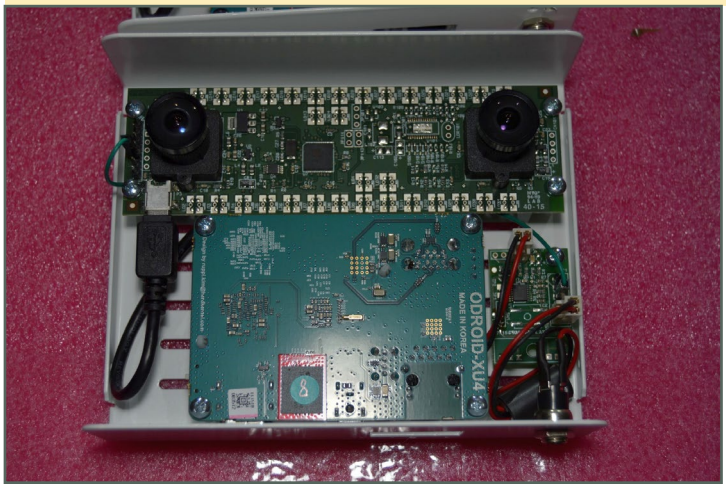
I have tested different ODROIDs: the X2, U2, U3, XU4, C1 and C2, all with good results, but I had the best feeling and experience with the U3. I am now working with the XU4, and apart from having a very good performance, I think the power consumption is excessive for our purposes. Although it is the model that we are using now, it may not be the best option. That's why we are currently testing the new C2 and trying to upgrade our devices to use it.

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

I've always missed having USB 2.0 connectivity in some of the pin connectors. Other things would be an internal WiFi, and on the C2 board, to have RTC on the device instead of through the RTC Shield module as it is now.

What hobbies and interests do you have apart from computers?

I like sports in general, especially football and Formula 1 racing. I still play football at Sant Celoni's veterans football club. I also like trekking with my family, to take advantage of the beautiful place where I live.



Martí built fully functioning stereos using an ODROID-X2 (above) and an ODROID-XU4 (below)

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

Beyond programming in assembler at University time, I have never taken any programming courses and have taught myself everything. Personally, what made me feel passionate about programming was to see that I wrote some sentences and the machine did exactly what I said. The first step is to feel interest and curiosity about programming, and the next step must be an internal desire to expand and improve the functionalities of the program. Continue searching and testing, and in the end it becomes easier. These days, the Internet puts everything in our hands to do just about anything. It's simply about searching, filtering, and applying until you find the key.