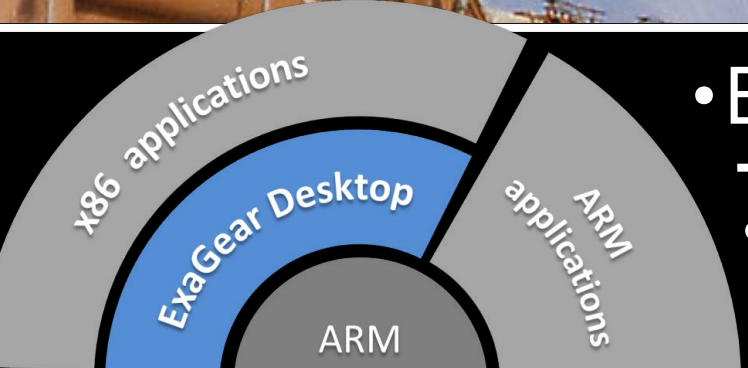
Half-Life on ODROID • Real Time Kernel • System memory with zRAM

ODROID Magazine

Year Three
Issue #27
Mar 2016

A new generation of ARM: 64-Bit Processing

The cutting edge ODROID-C2



- EXAGEAR Desktop for ODROID
- MQTT Basics: Learn how to enable IoT messaging across devices

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-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

EDITORIAL

The long-awaited 64-bit ODROID-C2 is finally here! Featuring 4K video, 2GB RAM, Gigabit ethernet, and a powerful S905 Amlogic processor, it offers cutting-edge power at a reasonable cost. It's available for $40 USD from the Hardkernel store at http://bit.ly/1fbE9Id. To learn more about the C2, and to download pre-built operating systems such as Android and Ubuntu, visit the new ODROID-C2 wiki page at http://bit.ly/1Trq5Ef. One of the exciting new peripherals available for the ODROID-C2 is the oCAM, an advanced USB 3.0 camera that can be used in OpenCV projects. DoYoon Kim details how to easily set up a simple hand tracking and surveillance system using open source software.

The popular ODROID-C0 can be used in conjunction with MQTT to create ioT devices, as shown by Venkat in his latest hardware tinkering article. Tobias continues his gaming series with Half Life, Andrew shows us how to use a real-time kernel to optimize your application's efficiency, Christopher presents a project on adding a backlight to the ODROID-VU7 display, Adrian helps us enhance our Linux terminal using Byobu, and Justin delves into Exagear to run Window programs on your ODROID.

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/1yplmXs.
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

Bruno went to Vegas to get married and enjoy his hard earned vacations. He enjoyed it a lot, but got back to our beloved magazine absolutely happy to see that our Hardkernel folks have released the ODROID-C2 into the wild, with great reviews. It finally giving him the opportunity to give the Gemini Rocket cover a go!

He also was super happy when he saw the guest list for his wedding ceremony, which listed Billy Corgan, Depeche Mode, the Victoria Secret model Candice Swanepoel and many other famous guests. But, in the end, it was just elaborate shenanigans from his sister-in-law that was kidding with him and his fiancée (now wife). In the end, the marriage was great and enjoyable.

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

I'm 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. I manage multiple accounts with agencies and filmmakers, from web design and programming, Analytics and Adwords, to video editing and DVD authoring, and help clients with the all aspects of online visibility. I own an ODROID-U2 and a number of ODROID-U3s, and loos forward to using the latest technologies for both personal and business endeavors. My 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.

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

ODROID-VU7 BACKLIGHT

ADDING A DIGITAL CONTROL SYSTEM

by Christopher Dean

Have you ever wanted to control the backlight on the ODROID-VU7 using the digital GPIO on the ODROID-C1? In this article, I will detail how to add a digital control for the backlight using only one transistor and two resistors. The digital switch can complement the existing hardware switch or bypass it completely.

Note that this is a semi-permanent modification and some soldering is required. Please be warned that, neither Hardkernel nor I would be responsible to rectify any mishaps resulting from this exercise. Also, the Hardkernel warranty would be null and void on the ODROID-VU as a result of following these steps, since the hardware is being modified. Figure 1 illustrates the completed circuit diagram schematic.

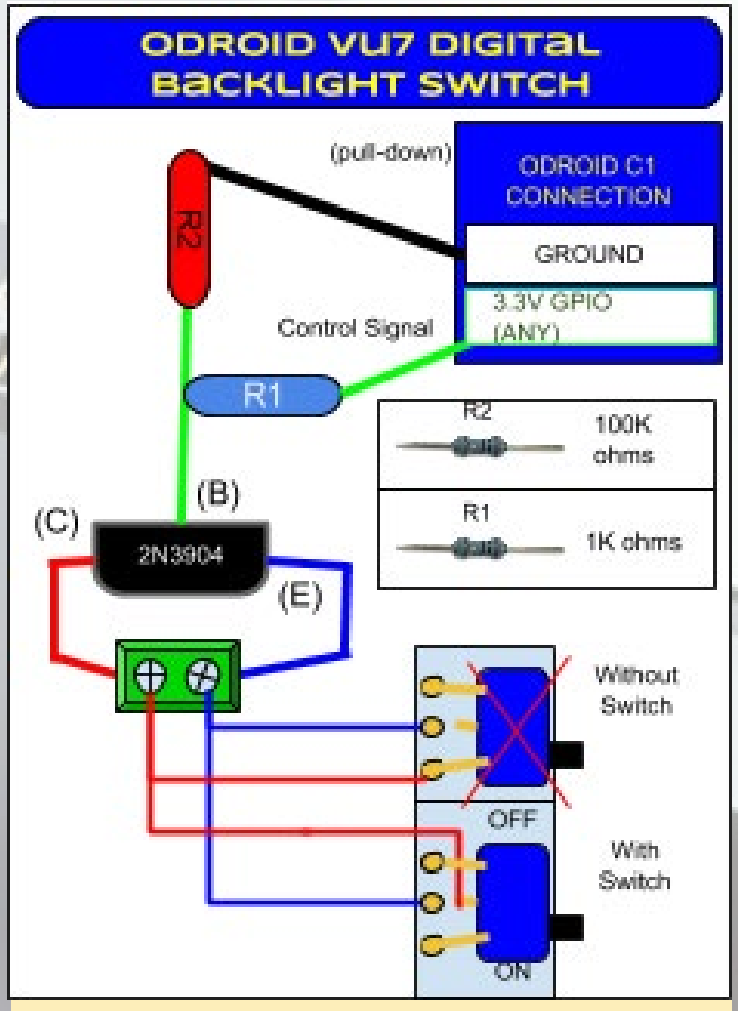


Figure I - ODROID-VU7 digital backlight switch schematic

Locate the "On/Off" backlight switch on the back of the display in the top left corner next to the 3 ports. Start by snipping the middle pin of the hardware switch. The cut should be somewhere near the middle. Create a complete disconnect by bending the snipped ends away from each other. The middle pin to be cut is shown in Figure 2.

Figure 2 - Middle pin to be cut, of the hardware switch

At this point, you need to decide whether you want to bypass the switch or retain full use of it. If use of the hardware switch is retained, the screen will turn off when the switch is in the "Off" physical position, irrespective of the state of the GPIO pin. If the switch is disabled, the digital control will be the only way to turn the display on or off. Make sure to follow standard safety procedures for yourself and the device during the soldering step.

In either case, solder one wire on the emitter side to the bottom half of the PCB middle pin (blue wire). To retain the use of the switch, solder the red wire (collector side) to the top

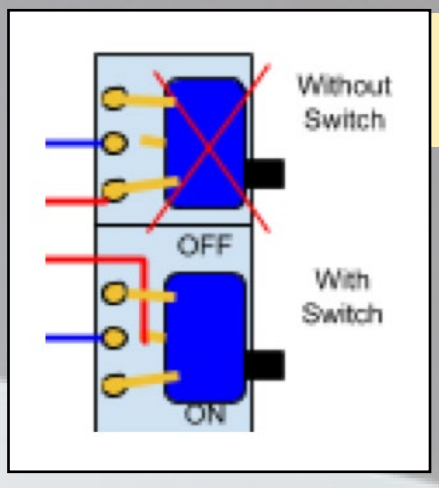


Figure 3 - Closeup of the hardware switch modification

half of the middle pin. On the other hand, to bypass the the hardware switch, solder the red wire (collector side) to the pin on the "On" side. Study figure 3 well before proceeding.

Figure 4 illustrates the state of the device for the case where the use of the switch is retained.

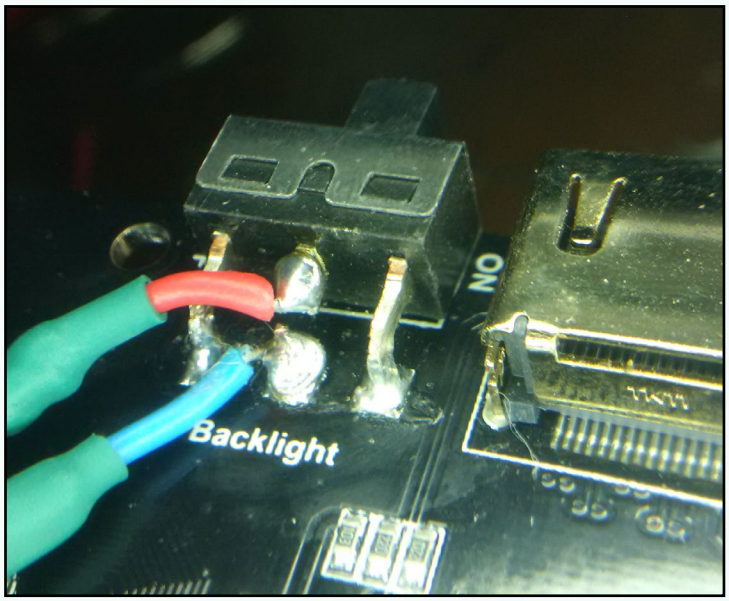


Figure 4 - Soldering is done to keep the switch

Using the illustrations and listed components, solder together the transistor (2n3904) and the resistors (1K & 100K ohm - ½ watt) on a proto-board. You can use a screw terminal as indicated in Figure 1, or solder the wires from the hardware switch directly to the transistor. Then, add a male or female header for the pull-down and signal pins, as appropriate to your use-case. Devote enough time to this step to ensure circuit accuracy and the absence of electrical shorts. It is recommended to wire-wrap and test the circuit before you solder it. Also, note that the transistor is likely to get very hot during use, so be sure to add an IC heatsink or another passive cooling option to prevent it from burning up. Figure 5 shows the result of steps performed so far.

Optionally, as the last step, mount the transistor board onto the back of the LCD panel and run the jumper wires to the C1. For testing, instead of toggling the transistor with the GPIO, you can also use pin 1 (3.3V power) to force the backlight to

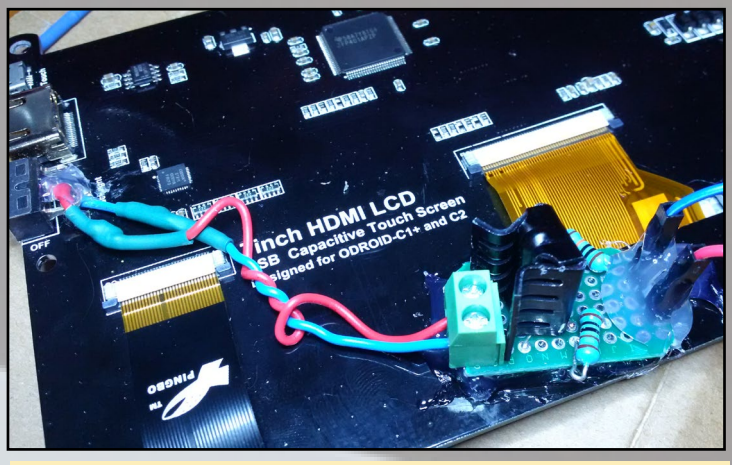


Figure 5 - Adding passive cooling

come on. Choose a GPIO pin and test using the pin to toggle the backlight. I used pin 7 (GPIO #83) in my test.

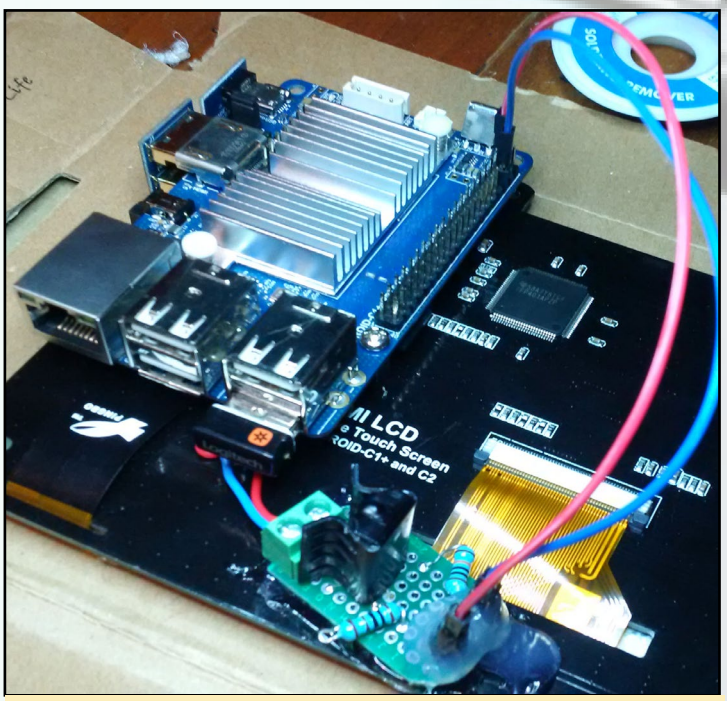


Figure 6 - Mount the transistor board onto the LCD panel

SUPER EYES

HAND TRACKING AND SURVEILLANCE WITH THE OCAM

by DoYoon Kim

An affordable, yet powerful camera module is now available! When you're looking for a camera module for an ODROID, you might be interested in a peripheral vision camera. However, it's almost certain that the price for such a camera will be higher than the price of the base development board. Naturally, the question that comes to many people is: why not just use a cheap webcam instead?

Sure you can, but a normal webcam has very limited abilities and lacks modularity. In other words, there is no flexibility or configurability for that simple webcam. First of all, one key thing that cheap webcams lack is the ability to change the camera's lens to suit your particular need. This is a large disadvantage, since the lens determines the field of view and the focal length. Because webcams are made for video chatting, they have a short focal length and a small field of view. This is because a camera only needs to be focused on your face for video chatting, which is normally located close to the camera. In many projects, you have to take a picture of an object that is far from the camera, or you may need to widen the field of view far beyond the range of a normal webcam. Furthermore, in a more serious application, you might need to have a camera with a shutter, however you will rarely find a webcam that supports this functionality.

Most webcams are made to be used with minimum hassle and configuration. This becomes a problem when need to control the camera's parameters, such as auto exposure. To process the image data properly, you need to keep the exposure time constant. However, webcams will typically have automatic modes, such as auto expose, that prevents manual manipulation and severely limits the camera's controllability.

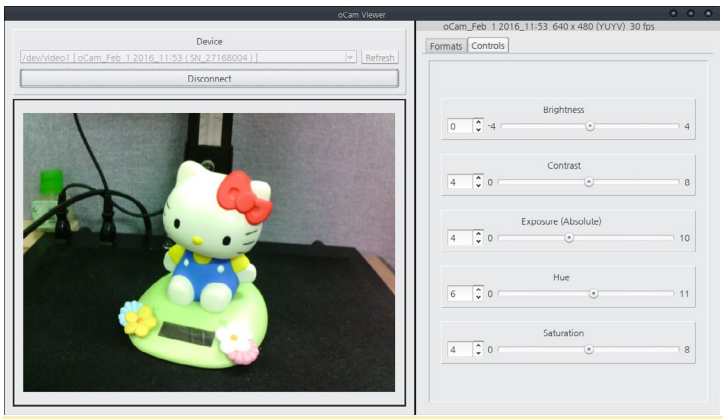
Another issue with most webcams is the interface between the camera module and the host board. One might think that, for most webcams, USB 2.0 interface is sufficient, and that there is no need to spend more money on a camera with a faster interface. Unfortunately, we encounter two major limits due to this slow data interface. The obvious limitation is on the image size and the frame rate which can be transmitted. Another hidden yet higher impact problem is with the CPU usage on the host board. Using a slow communication channel such as USB 2.0, it is inevitable that there will be compression and decompression of the video. At a frame size and rate of 1280 x 720 at 30 fps, a lot of CPU power is consumed.

The good news is that we finally have an alternative! A camera module for ODROID, named the "oCam", is available for purchase for USD $99 at `http://it.ly/1WsoNbr` and features the following:

M12 lens mount: The user can choose one of the M12 mount lenses that have 5 different focal lengths to use on their oCam. Optional parts will soon be available to adapt C or CS mount lenses.

Full controllability: The oCam comes with software that can control various camera parameters such as resolution and exposure time. The software's source code and binary executable for ODROID are available for download.

USB 3.0 interface: The oCam provides a high speed USB 3.0 interface which uses DMA to drastically lower the CPU utilization.



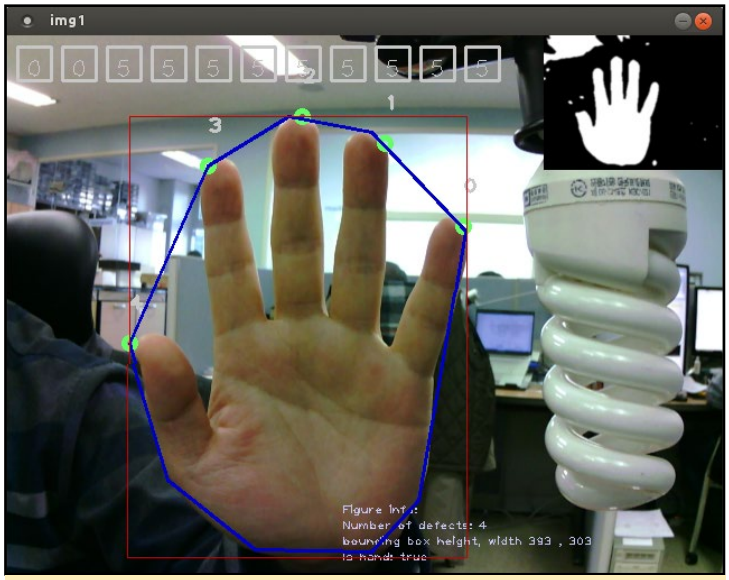
oCam Viewer to control camera parameters

It is available at an amazingly low price.

oCam is fully tested and compatible with UVC (USB Video Class), which does not require any device driver. This allows the oCam to run on Windows as well as Linux systems. Out of the many possible applications of the oCam, the following two examples best show you what is possible with the oCam.

Hand detection

This application was developed by Simen Andersen, and uses OpenCV to recognize and track a user's hand motion. SimenAndresen's blog post "Hand Tracking and Recognition With OpenCV" at `http://it.ly/1mOrrMu` contains more information.



Simen Andresen's handDetectionCV

You can run handDetectionCV on an ODROID-XU4 by following these steps:

Configure the ODROID-XU4 to access the Internet.

Open a terminal, enter the following command, then follow the prompts to install the OpenCV library:

```
$ sudo apt-get install libopencv-dev
```

Get the handDetectionDV source, move to the "Linux Version" folder, then build and run the program:

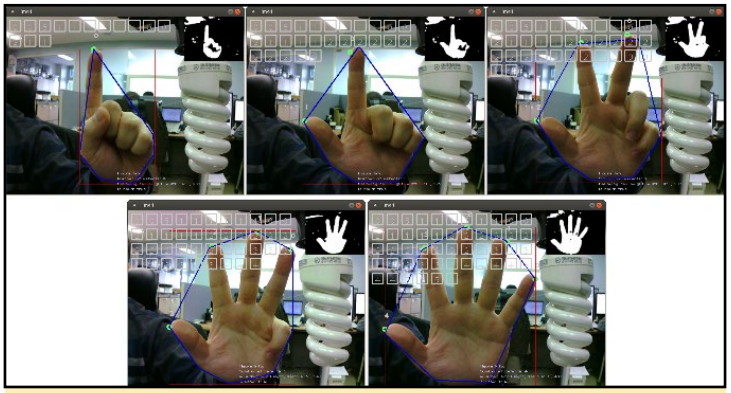
```
$ make all
$ opencv
```

Cover the green squares with one hand. Your hand has been tracked when the green squares turn into white squares as shown next. You can now move your hand around and see how the program tracks it.

Video Surveillance System

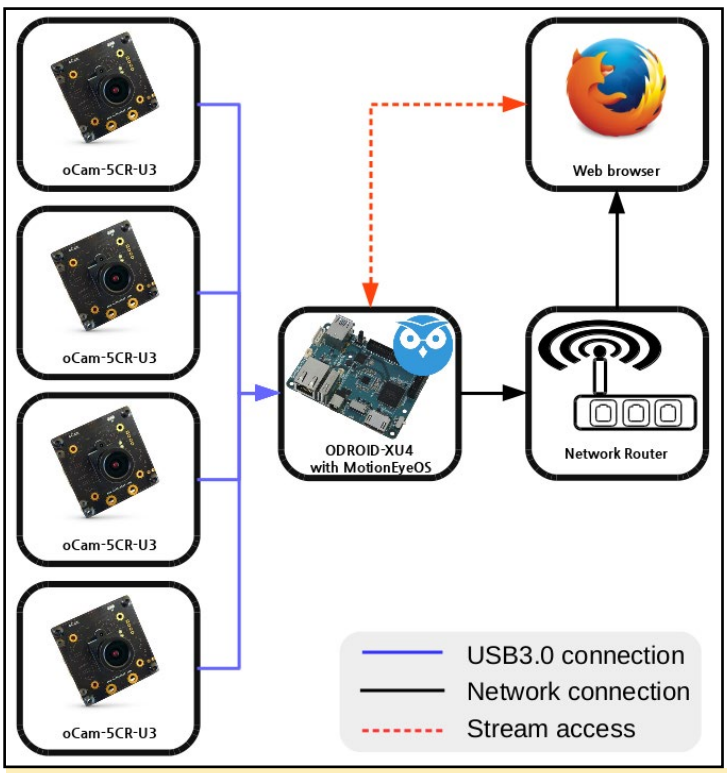
You can build your own video surveillance system using a single ODROID-XU4 and multiple oCam cameras. The surveillance system is based on MotionEyeOS, which was developed and published by Calin Crisan.

Hand detection by handDetectionCV



Hand tracking and recognition by handDetectionCV

MotionEyeOS GitHub: `bit.ly/1OlPlVE`
MotionEyeOS facebook: `on.fb.me/1ounjlP`



Video surveillance system configuration

To set up a surveillance system use the following steps:

• Prepare an ODROID-XU4 eMMC module or micro SD card and cameras. For the configuration in this example, we used 4 oCam cameras.

• Configure the ODROID-XU4 to access the Internet.

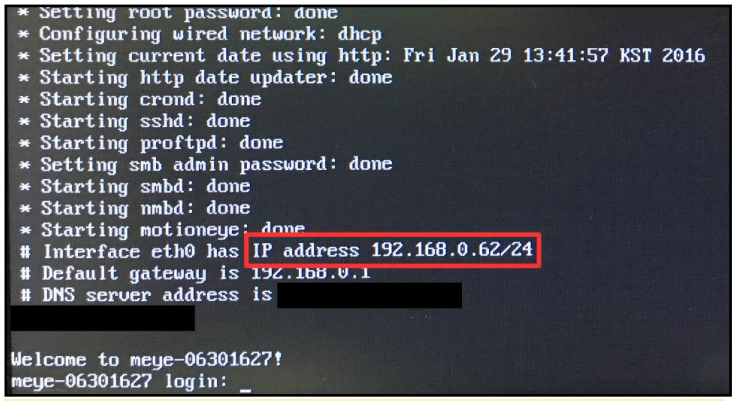
• Download the latest version of MotionEyeOS for the ODROID-XU4 from MotionEyeOS's GitHub "Supported Devices" page (`http://it.ly/214XmuP`) and unzip the downloaded OS file.

• Connect the eMMC module or micro SD card to your computer and flash the unzipped image file to it. If you're unfamiliar with this process, please read the ODROID flashing guide at `http://it.ly/1Vk9u4o`.

• Insert or attach the eMMC module or micro SD card loaded with MotionEyeOS into your ODROID-XU4.

• Using the Ethernet port, connect the ODROID-XU4 to a network and power on the ODROID.

Booting takes about 1 or 2 minutes. After this, you can log in to the system, using the default account of 'admin' with no password.

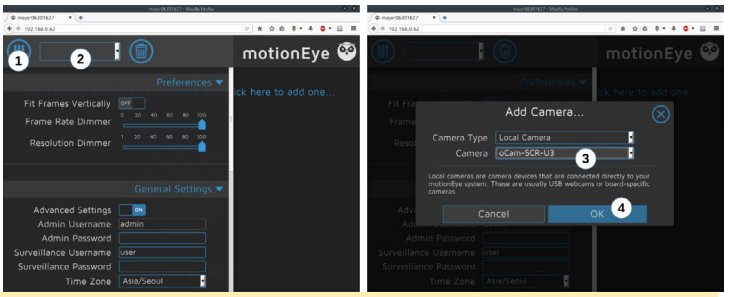


MotionEyeOS log-in screen

You can access the surveillance system from any web browser using the network address shown in the boot screen of MotionEyeOS.

Attach an oCam to ODROID-XU4.

From a web browser, log into MotionEyeOS using the admin account. Click "Add Camera" to add the oCam.



Adding oCam to MotionEyeOS system

Set the resolution to 640 x 480 and the framerate to 30fps, then finish by clicking "Apply".

Verify that the oCam's video appears. If you don't see any video coming from the oCam, try one of the following troubleshooting solutions:

Disconnect and reconnect the oCam. At the ODROID-XU4 console, check the connection to the camera using dmesg.

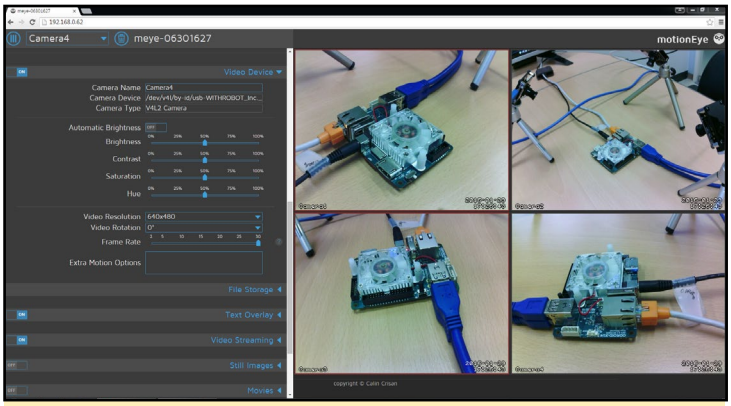
Add more power to oCam using either an external power supply or a powered USB 3.0 hub.

Connect the oCam to another Linux system to check if the oCam can transmit images at 640 x 480 resolution at 30fps correctly.

Repeat the connection process for the remaining unattached oCam cameras.

After completing those steps, you will now have your own surveillance system! The MotionEyeOS's monitoring screen will display something similar to the example shown below.

MotionEyeOS video surveillance system



MotionEyeOS video surveillance screen

Since MotionEyeOS is accessible through any web browser, you can view your camera's images and control them from anywhere as long as the ODROID-XU4 is connected to the Internet. Further information and more detail is available from MotionEyeOS's GitHub at `http://bit.ly/1VuPBHS`.

SHOW ME STUFF

A SMART KIOSK SYSTEM FOR YOUR ODROID-SHOW

by @matoking

Showmestuff is a spiffy usage for your ODROID-SHOW

A while back, I created a Python-based script called SHOWtime (http://bit.ly/1VfzMmW) in order to display a variety of interesting information on the ODROID-SHOW attached to a compatible ODROID. The information includes system statistics, website uptime tracker and Bitcoin price.

After upgrading to an ODROID-XU4, I created a web application using Flask and jQuery that essentially had the same features listed above, but in addition, is capable of interaction with the user. A video of the application in use can be seen at http://youtu.be/kVVemfqK-nI.

The application, including source code, installation and usage instructions, can be obtained from http://bit.ly/1TuJx2L. The application currently displays the following information through a slide-show/wizard like interface:

- **CPU & RAM usage,**
- **Disk usage,**
- **Tail, which displays the newest lines of a chosen file such as the kernel log,**
- **Bitcoin price, which comes with a fancy graph,**
- **Bitcoin address tracker, which displays current balance and most recent transactions,**
- **Website uptime tracker**

Note that the application has to be configured to use the VIEW_WIDTH and VIEW_HEIGHT parameters, based on your touchscreen's supported resolutions. The default should be correct for ODROID-VU and for the 5" and 7" Waveshare HDMI touchscreens.

After launching the web application, the user interface can be accessed through a web browser, by pointing it to the link: http://<ODROID-device-ip-address>:5210. If accessed at the device locally, you can use the link: http://localhost:5210 or http://127.0.0.1:5210. You can configure the views by updating the setting in settings.py, which is the web application's python source code file.

If you wish to add more views, you can do so by creating the following three categories of files:

Flask view (showmestuff/views/<your-view-name>.py): It will contain the HTTP requests required by the JavaScript application to retrieve the required information, such as CPU usage.

HTML view (showmestuff/templates/views/<your-view-name>.html): It will contain the HTML code that retrieves the view-specific configuration and the Javascript source code associated with the application.

JS file (showmestuff/static/js/views/<your-view-name>.js): Contains the app itself that retrieves information using the HTTP requests provided by the Flask view and displays them on-screen when the view is active.

For example, if you wish to add a view to display CPU usage, the phrase <your-view-name> listed above, could correspond to cpu-usage. You may use the uptime view as a simple example to develop and enable your own view. For comments, questions and enhancement suggestions, please visit the original post at http://bit.ly/1PGHNB8.

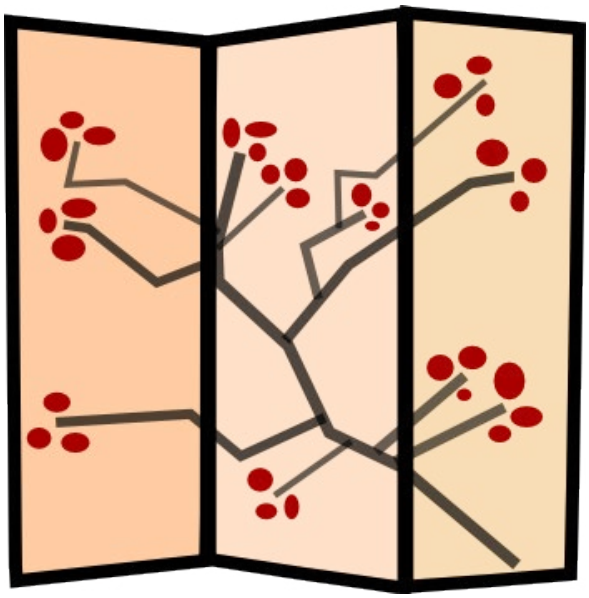
References:

http://bit.ly/1VfzMmW
http://flask.pocoo.org/
https://jquery.com/

BYOBU

YOUR LINUX TERMINAL ON STEROIDS

by Adrian Popa



Do you find yourself spending a lot of time in front of a terminal? Do most of your windows consist of terminals running half-forgotten command-line programs? Do you suffer from the pain of having your SSH disconnected either by timeout or a network failure and having to re-login and start your session over? Well, I think I have a solution for your problem—it's called byobu (http://byobu.co/). By the way, if you ever forget the name, search for japanese folding screens instead: (http://bit.ly/1Rm4Mlq)

Byobu is a text based window manager and a terminal multiplexer. It is designed to enhance tools like GNU Screen or TMUX. It includes enhanced profiles, convenient keybindings and toggleable system status notifications. I was drawn to it because of my need to see ODROID system temperature at all times while operating via SSH so as to prevent overheating.

In order to get started with byobu as quickly and easily as possible, the best thing to do is view their introductory video. (http://bit.ly/1QetGEI) It will show you what it can do and how to do it. This article describes some of the shortcuts and customizations you can use in the utility.

Getting Started

So, let's get started. To install byobu in Ubuntu:

```
$ sudo apt-get install byobu
```

To start it up, simply run byobu from your terminal:

```
$ byobu
```

and you get to your new terminal window. At first look it looks like a regular shell environment, but you will notice a status bar below showing you various system parameters and a slightly modified prompt, like in Figure 1.

The status bar has the following information enabled by default: Operating system and version, list of active terminals, un-updated packages, uptime, load, CPU scaling, memory usage and the current time and date. We will see in a minute how we can change them, but first, let's learn how to get around.

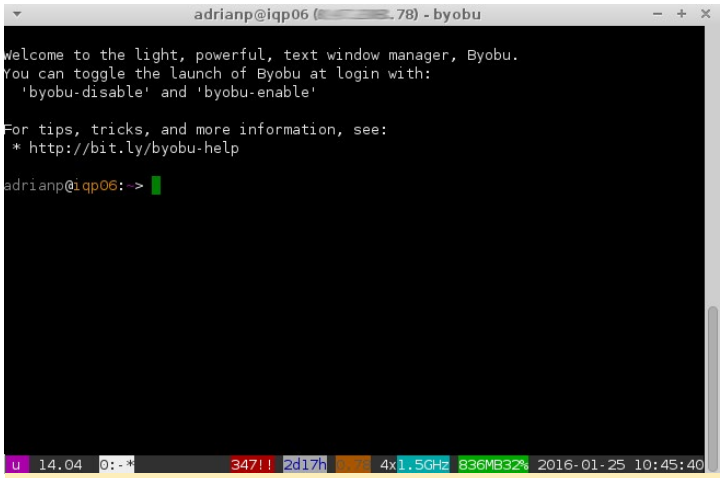


Figure 1 - Byobu main screen

Tips & Tricks

If you're not familiar with what screen or tmux can do, here's a quick refresher. Imagine having access to a single text-mode terminal, like a dumb terminal, but wanting to have the multitasking experience you're used to on a desktop environment. With screen or tmux you can start as many terminal instances you need (with CTRL+A C) and switch between them (CTRL+A followed by the terminal number) while having the advantage of keeping the terminals alive even if you disconnect from them (just replace CTRL+A with CTRL+B for tmux). Later, you can attach to any running tmux or screen session and resume your work. If you're interested in them, here's a comprehensive cheat-sheet (http://bit.ly/1QCPGuU).

Byobu aims to build on top of screen/tmux's capabilities and simplify general usage. To get byobu's key binding cheat

Figure 2 - Byobu cheat sheet

sheet, you only need to press Shift+F1 while byobu is running:

I suggest you take your time to play around and practice creating and switching windows, as well as practicing with splitting windows horizontally and vertically to better understand where and how you can best use the features available. You don't need to always keep splitting windows, but sometimes it can be useful to keep an eye on a log while editing a file or running some script.

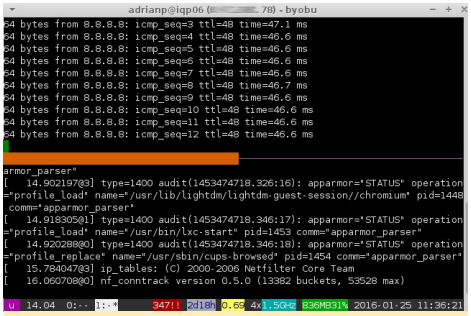


Figure 3 - Multitasking in the same window with splits

Since byobu makes a lot of use with function keys, it may conflict with other applications such as midnight-commander. To toggle byobu's key bindings, you can press Shift+F12. While its key bindings are disabled you can still use tmux or screen shortcuts to create or navigate between windows. When pressing CTRL+A the first time byobu

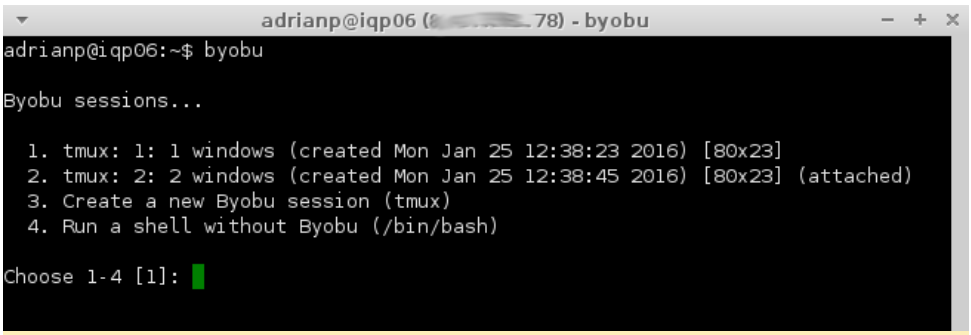


Figure 4 - Session selector in case of multiple sessions

Figure 5 - Cycling through status views

asks you if you want to turn on "screen" key-bindings or if you want the default Emacs line editing instead. You can change this at a later time by running:

```
$ byobu-ctrl-a
```

Now that you've played a bit with byobu, maybe you like it so much that you want it as a default shell when you log in. You can do this by invoking the menu (with F9) and selecting the last option, "Byobu currently does not launch at logon (toggle on)". Now the next time you log in, your bash startup scripts will start byobu up for you and you'll reconnect automatically to your session. If you have multiple sessions, you will be asked where you want to connect, like in Figure 4. You can connect from multiple places to the same session and the screen will resize to fit the smallest window. While this setting will not cause all new terminal windows under X11 to start up byobu, the Alt+F1 - Alt+F6 login shells will each start a new byobu session upon login—which might be useful for you.

Customizing the status bar

The status bar provides useful information about the health and performance of your system. By default there are some predefined "views" which you can toggle through with Shift+F5, as can be seen in figure 5.

To customize what is shown in the status bar, go to the menu (F9) and select, "Toggle status notifications" (Figure 6). If you enable "custom", you will be able to add your own scripts that can output useful information for you. You can read the man page to get an idea of what needs to be done in case you want to develop your own.

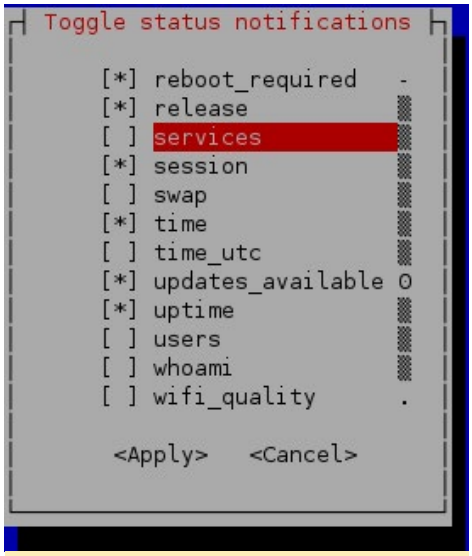


Figure 6 - Select desired plugins

For ODROID users I created a few small custom scripts that display system temperature and fan power percent relative to our own devices. You can download them from github (http://bit.ly/1KJh2gH) and install them with these commands:

```
$ mkdir -p . byobu/bin
```

```
$ sudo apt-get install bc
$ cd . /byobu/bin
$ wget https://raw. githubuser-
content. \ com/mad-ady/odroid-
byobu/master/20_fan
$ wget https://raw. githubuser-
content. \ com/mad-ady/odroid-
byobu/master/20_temperature
$ chmod a+x 10_*
```

The fan plugin should work on the ODROID-XU3 and ODROID-XU4, while the temperature plugin should work on all ODROIDs. The number prefixing the plugin's name refers to the frequency at which it's running, which in this example is every 20 seconds. Feel free to edit the plugins and set different thresholds to change the colors to better suit your needs. A sample output is in Figure 7.

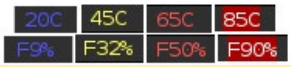


Figure 7 - Custom temperature and fan plugins

Troubleshooting and common problems

Here are some of the problems I ran into, and a few tips on solving them.

- When starting byobu each status update (every second) adds a new line at the bottom, stacking multiple updates.

This is caused by terminals not supporting UTF-8 (byobu's status line uses some UTF-8 encoded characters which messes up the row length if not properly supported by the terminal). To fix this, if you're connecting through putty try this fix (http://bit.ly/1mWHQ1r) or if you're SSHing from a different Linux system, try running byobu with the "C" locale:

```
$ LC_ALL=C byobu
```

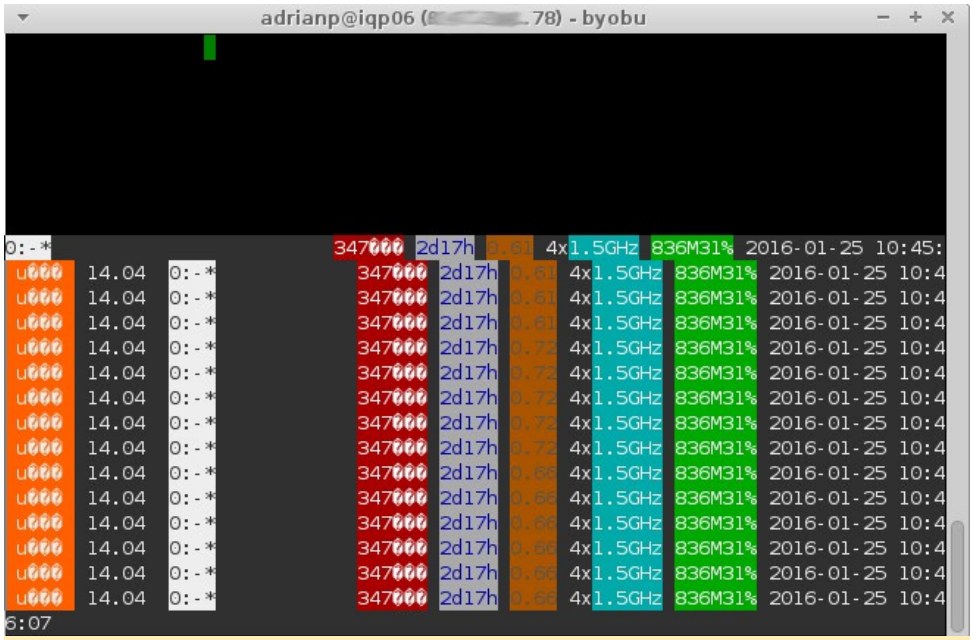


Figure 8. Byobu with broken UTF-8 support

- You may notice that scrolling back with your mouse wheel is broken.

By default tmux starts with mouse scrolling off. You can set it to on by running this command and restarting byobu:

```
$ echo 'set-window-option -g
mode-mouse on' >> . byobu/. tmux.
conf
```

Alternatively you can use the keyboard by entering "copy mode" with CTRL+A [and using arrows or PgUp/PgDown to scroll through the buffer.

- Help! I broke something and byobu crashes or freezes on startup and I can't login!

If something broke byobu or you want to login into a standard bash shell for example you can do this to bypass the automatic startup scripts:

```
$ ssh -t your-odroid-ip bash
```

- What happens when you're inside byobu and try to SSH into a byobu system?

Have you seen the movie Inception? By default byobu will detect you're running inside byobu (by the magic of environment variables) and will not connect you to a remote byobu session on the other end. You will just get a plain remote shell inside your local byobu. But if you want to, you can start up the remote byobu and experience the beauty of inception - but don't go too deep, you may get a headache… Thankfully you can run screen inside byobu as long as you use different key bindings.

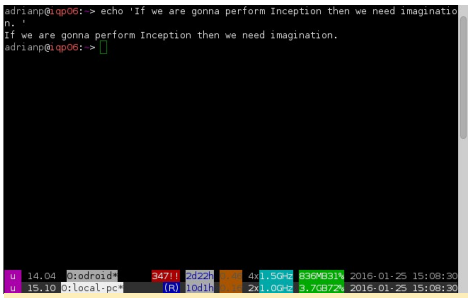


Figure 9 - The bottom byobu window runs on my PC while the inner window runs on the ODROID

See? With a little help and some elbow grease, the terminal world is not such a scary place after all!

HALF-LIFE

BLACK MESA HAS COME TO THE ODROID PLATFORM

by Tobias Schaaf

Welcome to Black Mesa! I've been waiting a long time to get this awesome game working on the ODROID platform. This is now possible, thanks to @ptitSeb, who ported Half-Life to ARM using the Xash3D engine and other libraries. By using of his version of GLshim, we can now play Half-Life in 1080p on ODROIDs!

This game is one of a kind, and made the First Person Shooter genre very popular. Similar to Doom's release many years earlier, this game had a major influence on the industry. There are countless mods for Half-Life, such as Counter Strike and Team Fortress. These games were originally just addons for Half-Life, but later on became their very own stand-alone games.

Figure 1 - Black Mesa Research Facility

Figure 2 - One of the Robots from the Intro of Half-Life

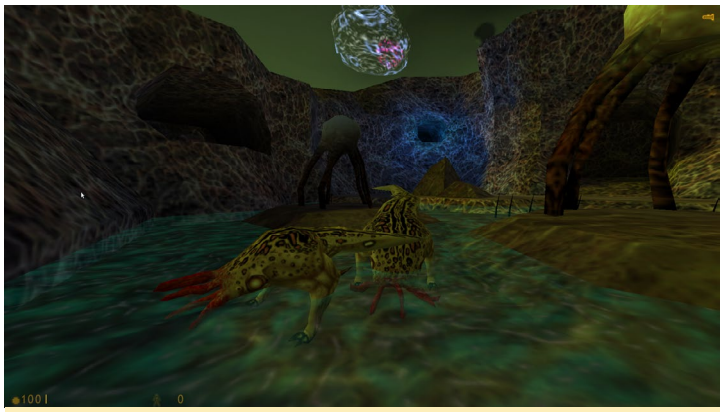


Figure 3 - Alien planet with the monsters that invade the Black Mesa Research Facility

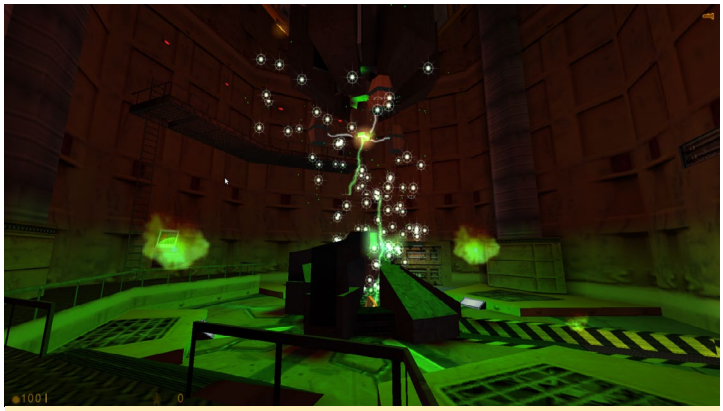


Figure 4 - Although it's already 18 years old, the game offers great visual effects

Using the following command, you can install the Xash3d engine from my repository using the Debian Jessie package list:

```
$ apt-get install xash3d-odroid
```

The game port offers support for single and multiplayer modes for the original Half-Life, as well as the Blue Shift addon. Other mods may work as well, such as Counter Strike.

To play, you will need version 1.1.1.0 of the original Half-Life. In your Linux home directory, there will be a folder named ".xash3d". Place Half-Life's "valve" folder inside of it. Half-Life mods will also need to be placed in that folder as well. See the following directory list as an example:

```
$ ll /home/odroid/.xash3d/
total 12
drwxr-xr-x  9 odroid odroid 1024
Feb  4 21:18 bshift
drwxr-xr-x 12 odroid odroid 1024
Feb  4 21:38 dmc
drwxr-xr-x 14 odroid odroid 3072
Feb  4 21:42 gearbox
drwxr-xr-x 12 odroid odroid 1024
Feb  4 21:42 ricochet
drwxr-xr-x 15 odroid odroid 3072
Feb  4 21:45 tfc
drwxr-xr-x 17 odroid odroid 3072
Feb  4 23:19 valve
```

For comments, questions, and suggestions, please visit the original article at http://bit.ly/1WsqDZF.

EXAGEAR

GET MORE FROM YOUR ODROID WITH TEAMVIEWER, SPOTIFY AND SKYPE

by Gaukhar Kambarbaeva

ELTECHS

ExaGear Desktop is a virtual machine that allows running of x86 Linux applications directly on ARM devices simultaneously with native applications. It also allows you to run x86 Windows applications on ARM platforms using Wine (http://bit.ly/1uHLDzo). The main advantage of ExaGear is its exceptional performance. The current version of ExaGear Desktop provides up to 80% of native ARM application performance on average, as described at http://bit.ly/20Ks9aK.

Although QEMU allows users to run x86 apps on ARM-based devices, it does not deliver good performance. A lot of x86 application developers do not take the time to do performance tuning of their applications, since they develop with an Intel i7 CPU. If their app works adequately on that machine, they do not tune performance further. ARM's CPUs have a less peak performance, but are more power efficient, cheaper and cooler, and with dramatic overhead of QEMU (5-10 times), you cannot run x86 apps at an acceptable performance level. ExaGear Desktop is a state-of-the-art emulator using binary translation, which provides small performance overhead.

One of the best ARM-based device for running x86 apps is the ODROID-XU4. Currently, it is the best device on the market from a performance perspective. It is based on an octacore Samsung Exynos 5422 2GHz CPU. By combining the high performance of the ODROID-XU4 and the efficiency of ExaGear Desktop, you can run a lot of popular x86 applications on an XU4 flawlessly and have a better user experience.

Additionally, ExaGear Desktop allows you interact with x86 applications the same way as you interact with ARM applications. After installation you can just go to the start menu and run your application. In this article, we will provide some real example of ExaGear performance by installing TeamViewer, Skype and Spotify.

Figure 1 - ExaGear Desktop shortcut on the desktop and x86 terminal

Installing ExaGear

First, you need to install ExaGear Desktop on your ODROID-XU4 device. You can purchase an ExaGear Desktop license key on the Eltechs website at http://bit.ly/1Q6SxKm. For the ODROID-XU4, you need to use "ExaGear Desktop for ARMv7". One license key is valid for one device for an unlimited time, with future updates of ExaGear Desktop available for free.

The most recent version of ExaGear Desktop is v1.4.1, which includes images of several x86 systems: Ubuntu 14.04, Ubuntu 15.04, Debian 7, Debian 8. There is an installation script that installs the x86 image on your host ARM system, so all that you need to do is to run the script in the directory containing the deb packages and the license key.

Since we will be installing a version of Spotify that currently only works on Ubuntu 14.04, we will install the guest x86 Ubuntu 14.04 image by running install script with following options:

```
$ sudo ./install-exagear.sh
ubuntu-1404
...
ExaGear is activated.
Done!
```

Once installation has been completed, click on ExaGear's shortcut on your desktop, or in the System Tools section of the Start Menu, and the terminal window with x86 system will open.

You are in x86 environment that can be checked by running the following command:

```
$ arch
i686
```

It is recommended to update the repositories on the first launch of the guest x86 system, after which you can install x86 application in the x86 terminal window:

```
$ sudo apt-get update
```

TeamViewer

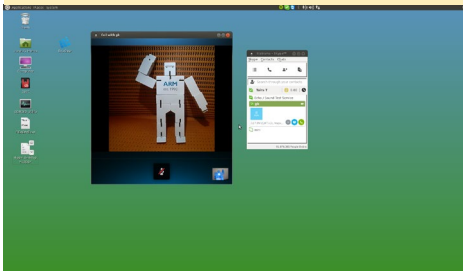
To install TeamViewer, run the following commands in the x86 terminal on your ARM device, just as you would on an x86 machine:

```
$ sudo apt-get install wget
$ wget http://download.teamview-
er.com/download/teamviewer_i386.
deb
$ sudo dpkg -i teamviewer_i386.
deb
```

During the execution of this command, you might see the following message:

```
Selecting previously unselected
package teamviewer.
(Reading database ...  7705 files
and directories currently in-
stalled.)
Preparing to unpack teamviewer_
i386.deb ...
Unpacking teamviewer (11.0.53191)
...
dpkg: dependency problems prevent
configuration of teamviewer:
 teamviewer depends on liba-
sound2; however:
  Package libasound2 is not in-
stalled.
 ...
 teamviewer depends on libxfixes3;
however:
  Package libxfixes3 is not in-
stalled.
dpkg: error processing package
teamviewer (--install):
 dependency problems - leaving
unconfigured
Errors were encountered while
```

Figure 3 - Skype with video



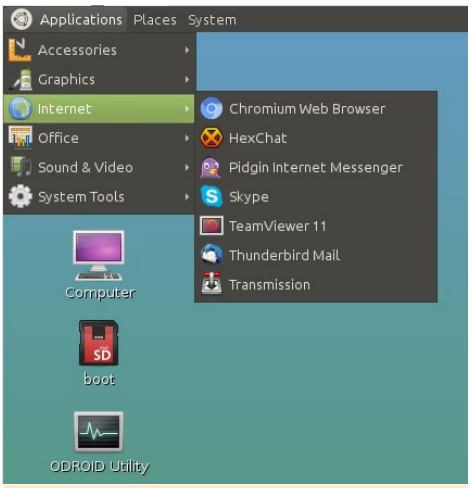


Figure 2 - TeamViewer and Skype in start Menu

```
processing:
 teamviewer
```

This message is only a warning informing you that the TeamViewer package depends on some other packages that should be installed. The following command will help you handle this situation by installing the dependency packages and finalizing the TeamViewer installation:

```
$ sudo apt-get install -f
```

Now you can run TeamViewer from the Start Menu, and use it as if it were running on x86 machine. Note that after a system reboot, TeamViewer will start automatically. If you would like to start or stop the TeamViewer daemon manually, make sure to do so from the x86 terminal.

Installing Skype

To install Skype, open the x86 terminal by launching ExaGear and running the following commands:

```
$ sudo apt-get install pulseaudio
$ wget http://download.skype.com/
linux/skype-debian_4.3.0.37-1_
i386.deb
$ sudo dpkg -i skype-debi-
an_4.3.0.37-1_i386.deb
$ sudo apt-get install -f
```

ODROID Magazine is on Reddit!

ODROID Talk Subreddit
http://www.reddit.com/r/odroid

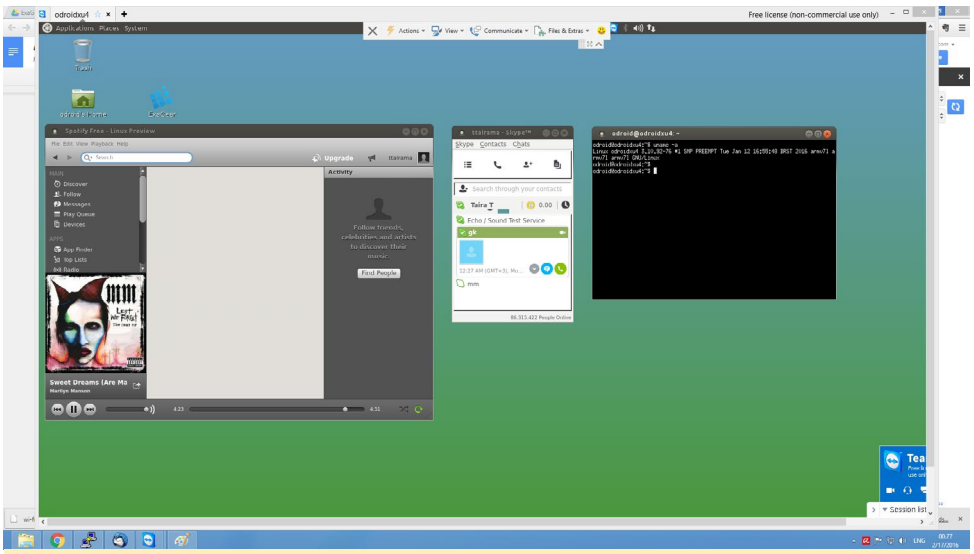


Figure 4 - Skype and Spotify running on an ODROID-XU4 connected to a Windows PC via TeamViewer

Once installation has completed, you can start Skype from the Start Menu, with text messaging, voice mail and video conferencing working flawlessly.

Installing Spotify

Installation instruction for Spotify is exactly the same as described on the official Spotify website at http://spoti.fi/1Wxngk8, except that you are running the commands on ARM device in the x86 terminal provided by ExaGear Desktop.

First, add the Spotify repository signing key in order to be able to verify the downloaded packages, add the repository, update the list of available packages, and install Spotify:

```
$ sudo apt-key adv --keyserver \
  hkp://keyserver.ubuntu.com:80
--recv-keys \
  BBEBDCB318AD50EC6865090613B00F
1FD2C19886
$ echo deb http://repository.
spotify.com \
  stable non-free | sudo tee \
  /etc/apt/sources.list.d/spo-
tify.list
$ sudo apt-get update
$ sudo apt-get install spotify-
client
```

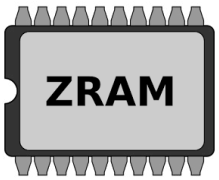
Now you can enjoy your favourit music!

Notes

Although the ARM ecosystem is growing rapidly, there are still many worthy applications that are not available on ARM. ExaGear Desktop makes possible to run these applications using an ARM device, and the combination of ExaGear Desktop and the ODROID-XU4 provides a nice experience.

USING ZRAM

MEMORY EXPANSION THROUGH COMPRESSION



by Tobias Schaaf

If you're comparing an ODROID to a conventional PC or laptop, one of the big differences is the amount of memory (RAM) that's available. While many computers come with 4-16GB of RAM or more, ODROIDs have only 2GB, and the C1 and C1+ have only 1GB of RAM. This can quickly lead to issues, and the programs that you're running can get very slow, and might even crash completely. There is even a chance that your entire system can crash. A way to avoid this behavior is to virtually increase the amount of memory available on the ODROID, using a technique called swap.

Swap

Swap is a file or a partition on your system that is used when your memory is full, in order to prevent a program or the system from crashing. Originally, swap was stored on a disk, which is, of course, much slower than real memory. Since the disk of an ODROID is an SD card or eMMC card, this means that it's even slower than on a conventional PC, which can use very fast solid state drives (SSD).

zRAM

You may have already heard about zRAM being used on ODROIDs to improve performance or to virtually increase the amount of RAM available to the ODROID. zRAM is a reserved space in memory that is used as a swap partition that can compress pages (data) in memory (RAM). Having the swap partition in memory is much faster than storing it on a hard drive or SD card. The only thing that needs to be done is to compress the data, which can be done very quickly by the CPU.

Classic swap vs zRAM

Memory is limited, so compressing data in memory is also limited. With a classic swap partition, you could theoretically have much more virtual RAM (swap) then with zRAM, but, as previously mentioned, this method a lot slower than zRAM. Therefore, if we use zRAM, we can have a very fast swap within certain limits. It's even possible to combine zRAM and regular swap. The zRAM is used first, and if you run out of zRAM swap space, the regular swap partition is used.

Overview

While using zRAM is a really good way to improve swapping of memory, it should only be done in moderation. If you read Internet articles about zRAM, you will find a lot of information, guides and scripts. One thing that they have in common is that they recommend using a maximum of 50% of the available memory as zRAM swap.

Because zRAM compresses pages in memory to between 30 and 50% of the original size, when you have 2GB of zRAM, that equals about 615 MB in normal memory. In practice, it's actually more since there is some administrative overhead as well. The ODROID-C1 has a total of 836 MB of available RAM, which means that only about 200MB would actually be available for the system to run on. The kernel, drivers, log files, foreground programs, and background tasks all need to run in that 200MB of memory.

It's important to set proper values for the size of the zRAM compression space. Whenever a program needs data from zRAM, it can't use it directly, but has to decompress the information first. If the RAM is already full, what needs to be done is to remove a portion of these 200-400 MB with active data from memory by compressing it again, and then decompressing the data that is needed for the current task.

With more data in the zRAM, the more often these swaps need to be done, which results in more stress on the CPU, since the CPU is constantly compressing and decompressing. In the end, the entire system will only be busy with compressing and uncompressing pages in the memory. The more memory you have to compress, as it approaches 200% of physical memory, the more likely that currently needed data is within the compressed pages.

Proper usage

No matter what kind of swap you're using (classic swap, zRAM partition, or both), swap is always a "last resort" measure of the system to prevent failures and crashes. It's not meant to be the default configuration of a system. If you system only runs with swap, your application design is flawed. If you want to run a program that requires at least 2GB of RAM, it's not a good idea to run it on a device that has less than 2GB total, or even less than 1GB as for the ODROID-C1. Even if you have classic swap or zRAM, that means you already know the program won't even run without using swap. The same approach applies to running many small programs simultaneously. If Chromium or FireFox needs 200 MB of RAM for each website that is open, launching 10 of them at a time is not a good idea either. Swap should only be used as a temporary measure.

Using zRAM

You can download the latest ver

sion of my zRAM scripts from http://bit.ly/1PHq51B and install them with dpkg:

```
$ wget http://bit.ly/1PHq51B \
  -O zram-odroid.deb
$ dpkg -i zram-odroid.deb # or
gdebi zram-odroid.deb
```

After installation, you can start and stop zram using the following commands:

```
$ sudo service zram start
$ sudo service zram stop
```

If you're using Ubuntu 14.04 or Debian Wheezy, you can also use the following command to get detailed information about your current use of zRAM:

```
$ sudo service zram status
```

If you're using a systemd based Linux (such as Ubuntu 15.04, 15.10, 16.04 or Debian Jessie), use the following command instead:

```
$ sudo zramstat
```

You can configure zRAM by editing the options in the zRAM configuration file located at /etc/zram/zram/conf.

Statistics

Fraction is the amount of RAM in % that should be used as zRAM. The default is 50, which means that 50% of available RAM should be used as zRAM. That's roughly 900 MB to 1GB on all devices except the C1, where it's only around 418 MB.

The value can be changed to nearly anything you want: 100 (same amount of zRAM as you have real memory), 200 (twice as much RAM as your system has can be used as zRAM) or even 2000 if you feel a little crazy. Good values for zRAM are probably between 50 and 100% of available memory.

Threads reports the number of working threads used to handle zRAM. The default is 0, which means there as many threads as there are CPU cores. This will create smaller zRAM devices under /dev/zram* equal to the amount of threads you selected. The amount of available zRAM will be evenly distributed among the zRAM devices. By default you have as many zRAM devices as you have CPU cores, but you might want to change this setting. If you're excessively using zRAM and also use all cores for zRAM, that means that in the worst case scenario, your system can't do any other tasks, since all cores are busy with zRAM. It's up to you to experiment with it, but you don't need to touch it if you don't want to.

Notes

Although zRAM can be compressed rather well to 30-50% of the original size, using large amounts of zRAM is not recommended. The more zRAM you use, the less memory (RAM) you have to actually work with, and the system can't work with zRAM directly but needs to uncompress it first. Which means that the smaller the amount of real RAM that you have available, the more often the system needs to compress and uncompress pages in zRAM. With large amounts of zRAM this happens way more often.

Depending on the compression level, 2GB of zRAM wouldn't even fit in the memory at all, causing application crashes or halting the entire operating system. Therefore, leaving enough "real" memory available for the system to work with is crucial, and you shouldn't set the zRAM value too high. As previously mentioned, values between 50-100% of your real memory are probably safe to use, and higher values are bound to cause issues over time.

Example

The C1 has a total of 836 MB of RAM while using a normal desktop image. Compressing the RAM to 30-50% of the original size mean that you would have a resulting size of 1672-2787 MB. As you can see, this already means that you may only get about 1.4 GB left over. Depending on the programs you run, a large amount of zRAM, such as 2GB, would not work and your memory would be depleted long before you reach the 2GB limit. Additionally, this would mean that 100% of your RAM would have to be compressed, but the system cannot work on compressed data. It needs to uncompress the data in memory in order to work with the data, and this has to be done in regular RAM.

Assuming we leave an absolute minimum of 136 MB, that leaves us with 700 MB of available RAM for compression. That means that we have 1400-2333 MB of zRAM for use, which also means that we have an absolute minimum of 136 MB to work with. All programs, drivers, the desktop itself, and kernel modules have to share 136 MB of memory in order to work. Every time a program needs data that's in the zRAM, a portion of that 136 MB of available memory needs to compressed and moved out of the way, and the data needed from zRAM needs to be uncompressed. With that small amount of RAM, your system would be constantly busy with compressing and uncompressing pages in memory, and your system gets slower and slower as the data stored in zRAM increases. Therefore, it's important to set the zRAM to an appropriate size, since it will slow down your system, and may create a situation where the system no longer reacts to input, since it's 100% busy with compressing and uncompressing.

Further reading

More information about zRAM may be found at http://bit.ly/1Pt80Gr.

MQTT BASICS

IOT MADE EASY

by Venkat Bommakanti

MQTT, which stands for MQ (Messaging Queue) Telemetry Transport, is an IBM invention, developed as a machine-to-machine (M2M) connectivity protocol. It was designed as an extremely lightweight publish/subscribe messaging transport, making it ideal for the Internet of Things (IoT) domain. This article introduces you to the basics of MQTT using the ODROID-C2, which is an ideal, powerful, next-generation IoT brain.

System preparation

Install the latest official Hardkernel ODROID-C2 image, then upgrade the system to bring in all updates using the following commands:

```
$ sudo apt-get update && \
  sudo apt-get upgrade
$ sudo apt-get dist-upgrade && \
  sudo apt-get install linux-
image-c2
$ sudo apt-get autoremove
```

Reboot the system and ensure that the device is functional, including access to the Internet.

Install pre-requisites

The ODROID-C2 does not include Real Time Clock (RTC) hardware support. However, an accurate clock is desirable when dealing with IoT related events. Towards that end, run the following command:

```
$ sudo apt-get install ntp
```

Then, update the system in order to synchronize the system clock to an NTP server after bootup and network connectivity is established. Update the /etc/rc.local file, which requires root privileges, add the following snippet, then eboot the system and ensure that the system clock is accurate:

```
( /etc/init.d/ntp stop
until ping -nq -c3 8.8.8.8; do
  echo "Waiting to setup ntp
based clock..."
done
ntpd -gq
/etc/init.d/ntp start )&
```

Please note that Hardkernel will be releasing an RTC hardware add-on for the ODROID-C2 in the near future. If that addon is used, the software fix noted above will not be required.

Mosquitto

Mosquitto is an open source Eclipse-project based message broker that implements the MQTT protocol. It includes a message publisher and subscriber test utilities that can be used to validate the installation.

Type the following command to install mosquitto and related software addons:

```
$ sudo apt-get install mosquitto
\
  apparmor mosquitto-clients
```

Check the installation using the following command:

```
$ mosquitto -h
mosquitto version 1.4.8 (build
date Fri, 19 Feb 2016 12:03:16
+0100)
mosquitto is an MQTT v3.1 broker.
Usage: mosquitto [-c config_file]
[-d] [-h] [-p port]
-c : specify the broker config
file.
-d : put the broker into the
background after starting.
-h : display this help.
-p : start the broker listening
on the specified port.
      Not recommended in conjunc-
tion with the -c option.
-v : verbose mode - enable all
logging types. This overrides
      any logging options given
in the config file.
See http://mosquitto.org/ for
more information.
```

Mosquitto 1.4.8 happens to be the latest available release. Launch the broker in background mode using the following command, which enables verbose logging:

```
$ mosquitto -d -v
```

Validate the launch using the following command:

```
$ ps aux | grep mos
mosquit+   431  0.0  0.1   6340
2348 ?        S      13:34   0:01
/usr/sbin/mosquitto -c /etc/mos-
quitto/mosquitto.conf
```

Test MQTT

First, create a text file to hold the message payload:

```
$ cd ~ && mkdir zBU && \
  cd zBU && mkdir mqtt && cd mqtt
$ touch pub.txt
```

Edit the file and add the following line, then save the payload file:

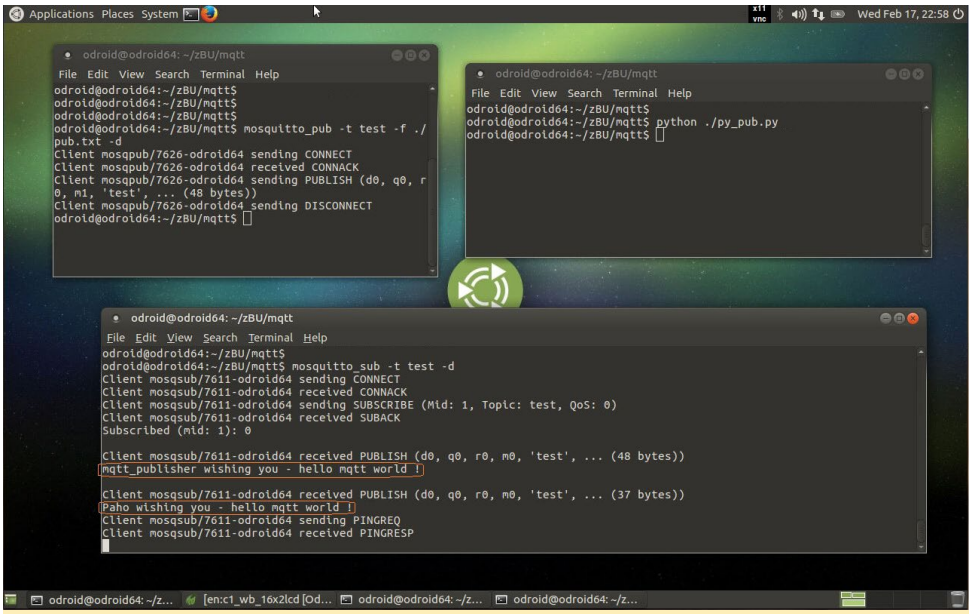


Figure 1: Message publishing and subscription using Mosquitto and Paho tools

```
mqtt_publisher wishing you -
hello mqtt world !
```

Launch the subscriber and publish a message using the publisher:

```
$ mosquitto_sub -t test -d
$ mosquitto_pub -t test -f ./pub.
txt -d
```

Note that the topic of "conversation" is test. Figure 1 illustrates the output in the terminal window where the subscriber was run.

Paho MQTT clients

Paho, another Eclipse-project, provides open source MQTT clients for various languages such as Java, C and python.

Install the paho python client using the following commands:

```
$ sudo apt-get install python-pip
$ sudo pip install paho-mqtt
```

Create a test python publisher using the following commands:

```
$ cd ~ && mkdir zBU && cd zBU &&
mkdir mqtt && cd mqtt
$ touch py_pub.py && chmod 755
py_pub.py
```

Edit the file and add the following contents, using the ODROID-C2 specifics:

```
#!/usr/bin/python

# Copyright (c) 2014 Roger Light
<roger@atchoo.org>
#
# All rights reserved. This pro-
gram and the accompanying materi-
als
# are made available under the
terms of the Eclipse Distribution
# License v1.0 which accompanies
this distribution.
#
# The Eclipse Distribution Li-
cense is available at
#    http://www.eclipse.org/org/
documents/edl-v10.php.
#
# Contributors:
#     Roger Light - initial
implementation

# This shows an example of using
the publish.single helper func-
tion.
#
# This is a modified version of
pub-single.py

# By Venkat Bommakanti

import sys
try:
      import paho.mqtt.publish as
publish
except ImportError:
      # This part is only re-
quired to run the example from
within the examples
      # directory when the module
itself is not installed.
      #
      # If you have the module
installed, just use "import paho.
mqtt.publish"
      import os
      import inspect
      cmd_subfolder = os.path.
realpath(os.path.abspath(os.
path.join(os.path.split(inspect.
getfile( inspect.currentframe() ))
[0],"../src")))
      if cmd_subfolder not in
sys.path:
      sys.path.insert(0, cmd_sub-
folder)
      import paho.mqtt.publish as
publish

publish.single("test", "Paho
wishing you - hello mqtt world
!", hostname="odroid64")
```

Then, run the python application:

```
$ python ./py_pub.py
```

A future MQTT article will address the publishing of ambient temperature, pressure and altitude data using the ODROID-Weatherboard on the ODROID-C2 and a Nore-RED solution.

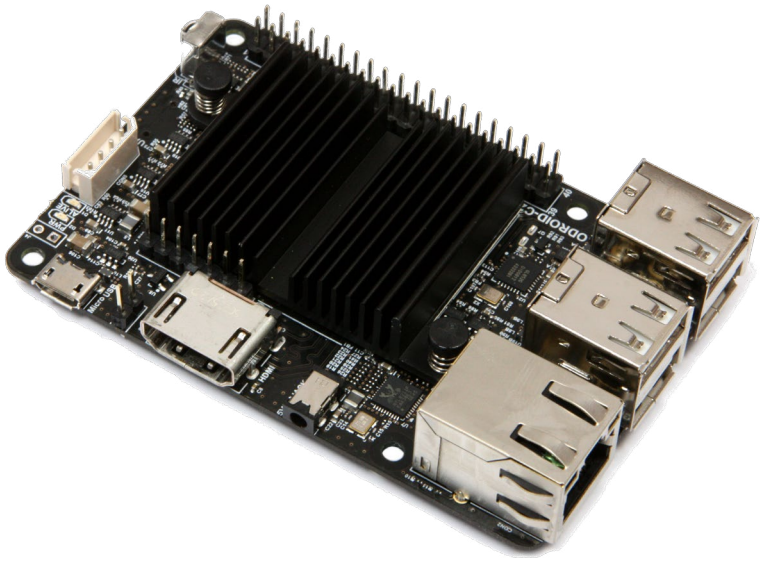
References

http://mqtt.org/
http://mosquitto.org/
http://bit.ly/24lowgd
http://bit.ly/1SLnQwL

ODROID-C2

64-BIT LOW-COST POWERHOUSE

by Justin Lee



The ODROID-C2 is esteemed to be the most powerful low-cost single board computer available, as well as being an extremely versatile device. Featuring a quad-core ARM 64-bit processor, advanced Mali GPU, and Gigabit Ethernet, it can function as a home theater set-top box, a general purpose computer for web browsing, gaming and socializing, a compact tool for college or office work, a prototyping device for hardware tinkering, a controller for home automation, a workstation for software development, and much more. It is available for purchase at the Hardkernel store (http://bit.ly/1Pp4J7w) for USD$40.

Some of the modern operating systems that run on the ODROID-C2 are Ubuntu, Android, ARCHLinux, and Debian, with thousands of free open-source software packages available. The ODROID-C2 is an ARM device -- the most widely used architecture for mobile devices and embedded 64-bit computing. The ARM processor's small size, reduced complexity and low power consumption makes it very suitable for miniaturized devices such as wearables and embedded controllers.

Specifications

- Amlogic ARM® Cortex®-A53(ARMv8) 2Ghz quad core CPUs
- Mali™-450 MP3 GPU (OpenGL ES 2.0/1.1 enabled for Linux and Android)
- H.265 4K/60FPS and H.264 4K/30FPS capable VPU
- 2GB DDR3 SDRAM
- HDMI 2.0 4K/60Hz display
- Gigabit Ethernet
- 40pin GPIOs + 7pin I2S port
- eMMC5.0 HS400 Flash Storage slot / UHS-1 SDR50 MicroSD Card slot
- USB 2.0 Host x 4, USB OTG x 1 (power + data capable)
- Infrared(IR) Receiver
- Ubuntu 16.04 and Android 5.1 Lollipop based on Kernel 3.14 LTS
- Board dimensions are identical to the ODROID-C1+

Market comparison

All of the boards compared in Table 1 are Linux-friendly ARM® single-board computers that cost less than USD$40.

Hardware

The ODROID-C2 has many advantages over the Raspberry Pi 2. The processor is an S905 2GHz quad-core from Amlogic with 2GB DDR3 RAM, Gigabit Ethernet and IR-re-

Table I - ODROID-C2 vs ODROID-CI vs Raspberry Pi2

	ODROID-C2	ODROID-C1	RPi 2 Model B
CPU	Amlogic S905 SoC 4 x ARM® Cortex®-A53 2GHz ARMv8 Architecture @28nm	Amlogic S805 SoC 4 x ARM® Cortex®-A5 1.5GHz ARMv7 Architecture @28nm	Broadcom BCM2836 4 x ARM® Cortex®-A7 900Mhz ARMv7 Architecture @40nm
GPU	3 x ARM® Mali™-450MP 700MHz	2 x ARM® Mali™-450MP 600MHz	1 x VideoCore IV 250MHz
RAM	2GB 32bit DDR3 912MHz	1GB 32bit DDR3 792MHz	1GB 32bit LP-DDR2 400MHz
Flash Storage	Micro-SD UHS-1 @83Mhz/SDR50 or eMMC5.0 storage option	Micro-SD UHS-1 @78Mhz/SDR50 or eMMC4.5 storage option	Micro-SD @50Mhz/SDR25, No eMMC storage option
USB2.0 Host	4 Ports	4 Ports	4 Ports
USB2.0 Device / OTG	1 Port for Linux USB Gadget driver	1 Port for Linux USB Gadget driver	No
Ethernet/LAN	10/100/1000 Mbit/s	10/100/1000 Mbit/s	10/100 Mbit/s
Video Output	HDMI 2.0 4K/60Hz	HDMI	HDMI 1.4 / RCA /DSI
Audio Output	HDMI / I2S	HDMI	HDMI / 3.5mm Jack
Camera Input	USB 720p	USB 720p	MIPI CSI 1080p
Real Time Clock	No(unless using an add-on module)	YES (On-board RTC)	No(unless using an add-on module)
IR Receiver	YES (On-board IR Sensor)	YES (On-board IR Sensor)	No(unless using an add-on module)
IO Expansion	40pin port (GPIO/UART/I2C/ADC) 7pin port (I2S)	40pin port (GPIO/UART/SPI/I2C/ADC) 7pin port (I2S) : ODROID-C1+ only	40pin port (GPIO/UART/SPI/I2C/I2S)
ADC	10bit SAR 2 channels	10bit SAR 2 channels	No (unless using an add-on board)
Size	85 x 56mm (3.35" x 2.2")	85 x 56mm (3.35" x 2.2")	85 x 56mm (3.35" x 2.2")
Weight	40g (1.41 oz)	40g (1.41 oz)	42g (1.48 oz)
Price	$44	$35	$35

ceiver. The size of this computer is still only 85 x 56 mm with a weight of 40g, and offers silent operation, 2~5W average power usage, and instant portability, since it fits in a shirt pocket.

One powerful feature of the ODROID-C2 is the row of GPIO (general purpose input/output) pins along the edge of the device. These pins are a physical interface between the board and the outside world. The 40pin interface header includes PWM, I2C, UART, ADC and GPIO function.

An SD 3.01 standard compatible UHS-1 MicroSD card, as well as the faster eMMC module, can be ordered with the ODROID-C2, and arrives with the popular Ubuntu operating system already installed. Insert the SD card into the slot, connect a monitor, a keyboard, a mouse, Ethernet and power cable, and that's all you need to do to use the ODROID-C2! Browse the web, play games, run office programs, edit photos, develop software, and watch videos right away.

The IR receiver and ADC features on the ODROID-C2 offer many options for building great DIY projects.

CPU/RAM performance

We ran several benchmarks to measure the computing power on the C2. The same tests were performed on the Raspberry Pi 2, ODROID-C1, ODROID-U3 and ODROID-XU4. The values of the test results were scaled uniformly for comparison purposes. The computing power of the C2 was measured to be ~2-3 times faster than the latest Raspberry Pi 2 thanks to the 2Ghz Cortex-A53 cores and much higher memory bandwidth. The high-performance 2GB DDR3 RAM is an additional advantage allowing most programs to run smoothly on the C2.

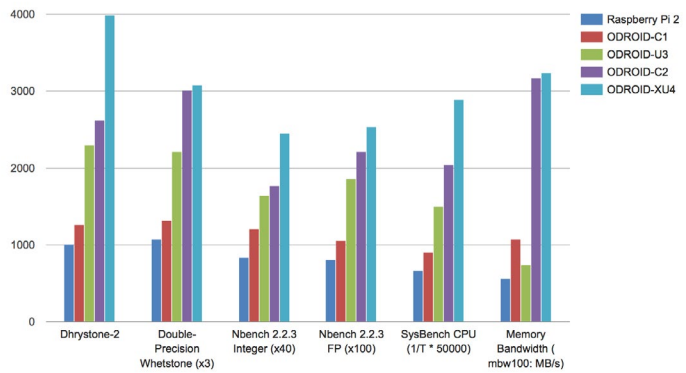


Figure I - CPU/RAM benchmarks graph

Storage performance

The C2 can boot from a MicroSD card or an eMMC module, which is selected using an easy-access hardware switch. The MicroSD interface supports the higher performance UHS-1 mode as well. File access of a 512MB file (read/write) on two different storage options shows distinct performance differences.

The eMMC 5.0 storage is ~7x faster than the MicroSD Class-10 card in read tests. The MicroSD UHS-1 card is ~2x faster than the MicroSD Class-10 card in read tests. The MicroSD UHS-1 card provides a great low-cost option for many applications! But don't forget that the new eMMC Black modules are also very affordable, thanks to the new price cut.

Ethernet performance

The C2 has an on-board Gigabit Ethernet controller. Our bi-directional streaming speed was measured at ~900Mbps. Thanks to the doubled Tx buffer in S905, the upload speed is twice faster than C1.

Display

As shown in the screenshot below, the 4K HDMI output renders a gorgeous desktop screen at ultra-high-definition 3840x2160 resolution, ready for the new generation of televisions and monitors.

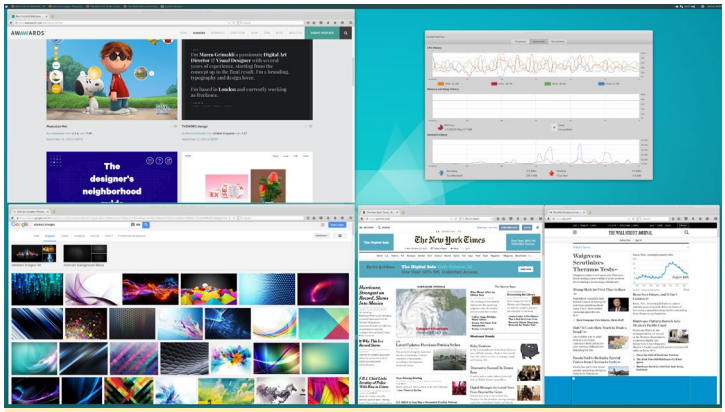
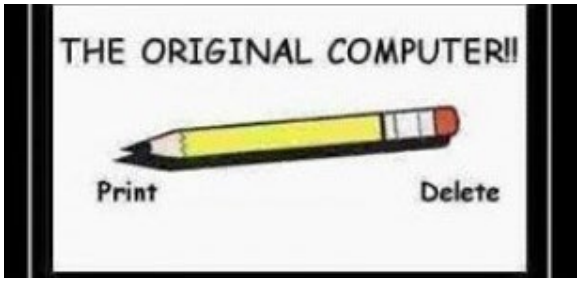


Figure 2 - 4K desktop screenshot

Notes

There is no Serial Peripheral Interface (SPI) bus or on-board Real Time Clock (RTC) on the C2, since the S905 SoC doesn't offer them. As a results, many SPI-based add-on boards are not compatible with C2. However, we are considering making an add-on RTC board. Also, the alpha-blending issue has been fixed in the S905, so we don't need to use the DDX blending any more.

OS images and build guides are available in our WiKi at http://bit.ly/1Trq5Ef.



REAL TIME LINUX KERNEL

THE ODROID-CO'S BEST FRIEND

by Andrew Ruggeri

Recently, ODROID forum user @chlorisdroid (http://bit.ly/1PIuRIR) posted a preemptive Real Time (RT) Linux 3.10 kernel for the ODROID-C0, C1, and C1+. The forum thread, available at bit.ly/1QAF0af, features GitHub links to the modified kernel source and related infomation. With the recent release of the ODROID-C0, which is a development board aimed at the Internet of Things (IoT) and portable use, a real-time Linux kernel offers a great advantage.

This article will detail what exactly a real time Linux kernel is. I hope that when you're finished reading it, you will be as excited as I am for all the advantages that an RT Kernel can bring to the ODROID-C0, C1, andC1+. You may be wondering, if a real time Linux kernel is so great, why isn't it the default Linux kernel that Hardkernel provides? The simple answer is that although both standard and real time kernels are very similar, they serve different uses.

What's in a name

The first step to understanding what this modified real time kernel offers is to understand its name. The full title used on the forum post to describe the kernel was "Real Time Linux Kernel 3-10-80-rt88 PREEMPT_RT". The main part of interest is at the end "PREEMPT_RT". The numbers in the middle are the version information.

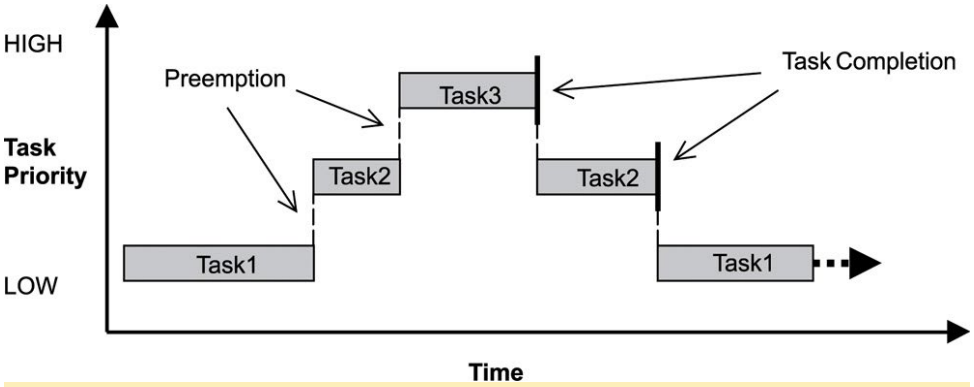


Figure I: Spinlock of a low priority task

"PREEMPT_RT" stands for Preemptive Real Time. The standard Linux kernel after version 2.6 is partially preemptive by default. A kernel that is preemptive has the ability to change the task that the CPU is focused on. The CPU can be working on one task and then be told to switch its focus temporarily to another task. Once that other task is completed, focus is restored to the previous task. One task being switched for another is called context switching. Tasks are given a priority which is used to determine which task is more important and should get the CPU's focus. A low priority task, such as background wifi scanning, will get interrupted frequently as higher priority tasks come up. Conversely, a task with high priority will get much more time from the CPU.

One important note is that a task can hold its place with the use of a spinlock. This means even if a higher priority task comes along, the task with the spin-locked task will remain in focus. Figure 1, courtesy of embeddedlinux.org.cn, shows an example of a low priority task (Task 1), which gets interrupted by a higher priority task (Task 2). Task 2 is then overtaken by a high priority task (Task 3). Once the higher priority task (Task 3) is completed, the next highest task takes control (Task 2). The same situation is seen between Task 2 and Task 3.

Real Time is the most important part of the kernel's description, which sets it apart from a normal kernel. A Real Time Operating System (RTOS) is a system that uses time constraints and deadlines for task completion. The main use for an RTOS is to be deterministic, which means that the OS will do what you want at the time that you want it to happen. However, keep in mind this does not mean that an RTOS is faster, or that it will have better throughput at task completion than a non-RTOS. In fact, the opposite is true.

Steven Rostedt, a maintainer of the

Preemptive Real Time Patch, is often quoted as writing "The more determinism you have, the less throughput you have" with regard to a Real Time System. Figure 2, courtesy of Altera (http://bit.ly/1Xu3Vl3), shows the jitter distribution by comparing 2 kernels, one with the RT patch and one without. Figure 3, provided by Red Hat (http://red.ht/24b4oNN), shows the repsonse time of a given task between an RT and a non-RT kernel. The statistics on Figure 2 show that the RT kernel has a significantly lower skew and standard deviation. The small standard deviation is what makes the RT kernel of interest, as this shows that timing is very consistent.

Figure 3 also shows the consistency of Real Time kernel latency, where the non-RT kernel (in red) has a range of about 175 microseconds, and the RT tasks (in green) have a more limited range of about 20 microseconds. When looking at the typical latency in Figure 3, the non-RT times are commonly lower as compared to the RT's times. This is expected since, as previously mentioned, this is a drawback to the increase in determinism.

In a RTOS, a task is given an allotment of time and its processing should be completed within that allotment. There are two terms commonly used to refer to how a processes behave in a RTOS: hard, and soft. Hard real time means if a task does not finish within its deadline then major problems or complete failure will or have occur. With Soft Real Time if a task is not completed within its time constraint minor or negligible problems will occur. The Linux kernel, after 2.6, is only setup to handle soft deadlines, The PREEMPT_RT patch adds the ability for hard deadlines as well.

Figure 2 - Jitter distribution between an RT and a non-RT kernel

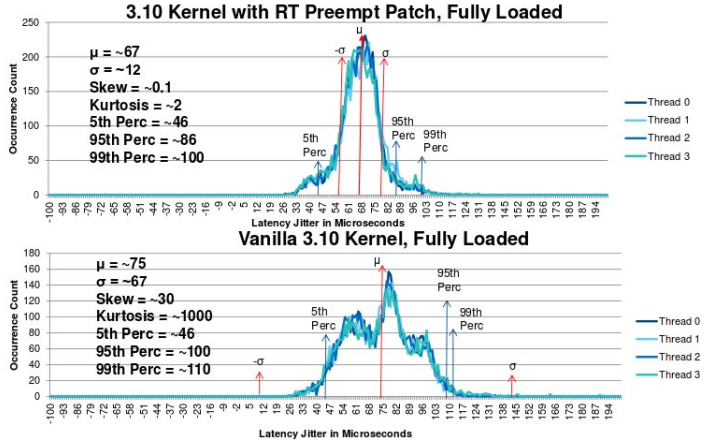
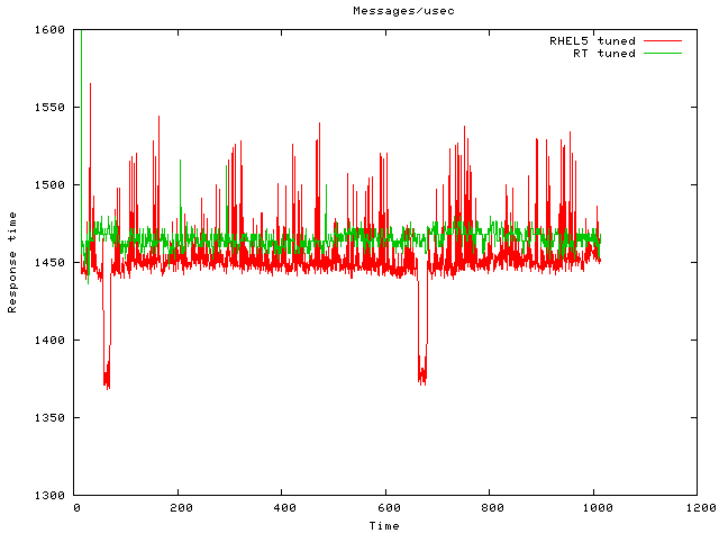


Figure 3 - Latency between an RT and a non-RT kernel



How it works

The PREEMPT_RT patch works by changing the behavior of a few key elements. The patch aims to be fully preemptive, beyond what the normal kernel considers preemptive. This means that the spinlocks, which can stop a context switch, do not make sense if the goal is to be fully preemptive. Spinlocks in the RT kernel are treated as mutexes instead, and so lose their ability to lock a task. Additionally, critical sections marked with rwlock_t and spinlock_t are also not preemptive. The final major change is to the behavior of interrupts, which run as threads when fired. More technical details on the functionality of the RT patch can be found in the links at the end of this article.

The cleanest example of when to use an RTOS comes from Abraham Silberschatz in his book Operation System Concepts: "A real-time system is used when rigid time requirements have been placed on the operation of a processor or the flow of data". With that in mind, it's easy to see why an RTOS can be found in such a variety of places and device, such as aircrafts, trains, small MP3 players, and quadcopters. Any application where you might find the need for time sensitive tasks is a candidate for using an RTOS.

Further reading

2013 presentation at the ELC by Steven Rostedt
http://bit.ly/1Lqz8Pl

Real Time Linux Wiki
http://bit.ly/1OQlDcv

@chlorisdroid's blog
http://blog.georgmill.de

@chlorisdroid's Real Time Linux forum post
http://bit.ly/1QAF0af

TONER RESET

EXTENDING THE LIFE OF YOUR LASER PRINTER CARTRIDGES

by @Jojo

I created a micro-project for resetting the counter of remaining toner for my Samsung CLX-3175N laser printer. Some Samsung laser printers are shipped with pre-filled starter toner cartridges. These cartridges do not have an individual counter for pages and toner, so the printer counts these values by itself, as long as a cartridges with individual counter chips is not being used.

However, this is a very bad business practice of Samsung: for each cartridge, they assume how much toner is used, and how many pages can be printed with the remainder. However, if the printer believes that there is no toner left, it will refuse to print, no matter how much toner is actually left.

However, many Samsung printer models have internal EEPROMS for storing information, such as the usage values of all internal components. By determining which EEPROM addresses the printer uses to store certain information, it's possible to reset the information, which is the goal of my project.

Preliminary design

I wanted to have the reset process be as convenient as possible. To achieve this, I decided to remove the internal SMD EEPROM and replace it with an external one in DIP. I desoldered the EEPROM off the printer's main PCB and soldered wires to the main board's EEPROM pads. The wires reach into an externally accessible place inside my printer.

Then, I made a PCB with a socket for the new DIP EEPROM and a microcontroller, because I wanted to hook up the microcontroller to the EEPROM for occasionally resetting the toner values. After the PCB was finished, I realized that I am too lazy to write the microcontroller's program, so I needed another idea.

Second approach

After extracting the printer's EEPROM, I needed a convenient way to read and write to it, which is when I thought of using my beloved ODROID-C1. After some Internet research, I found that it is quite easy to hook up an EEPROM to an SBC like the C1. I found an article written by someone who did basically the same thing as I was planning to do, so I made another small PCB that connected an EEPROM to my C1. At the moment, this process is not as convenient as it could be, because I need to switch the EEPROM between my printer and my C1, but it takes less than a minute to do so.

I wrote an automatic script that creates a backup of the EEPROM with time stamp, then resets the values for the toner count and remaining toner. In order to make the script work for you, first install the dialog application, because I love fancy DOS-like windows:

```
$ sudo apt-get install dialog
```

Then, install a tool to scan the I2C bus:

```
$ sudo apt-get install i2c-tools
```

You can optionally install a hex editor for doing further research:

```
$ sudo apt-get install hexedit
```

Next, download, compile and install the eeprog tool. There is a modified version of the program that has to be used, because the modified version supports a "delay" option when performing a write operation. Download the file from http://bit.ly/1OkTf0Z and extract it to your home directory. Then, navigate to the directory where you extracted the program and compile it to create an executable program:

```
$ cd ~/eeprog-0.7.6-tear12
$ make
```

Load the I2C module:

```
$ sudo modprobe aml_i2c
$ sudo echo "aml_i2c" >> /etc/
modules
```

After the module has loaded, hook up the EEPROM to the I2C bus of your C1, which should be detected at address 0x50 by running the following command:

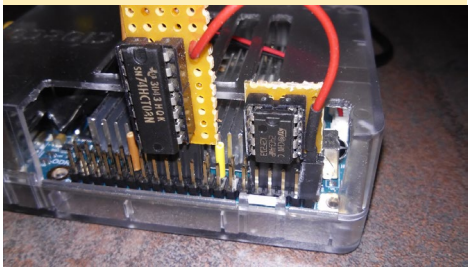
```
$ sudo i2cdetect -y 1
```

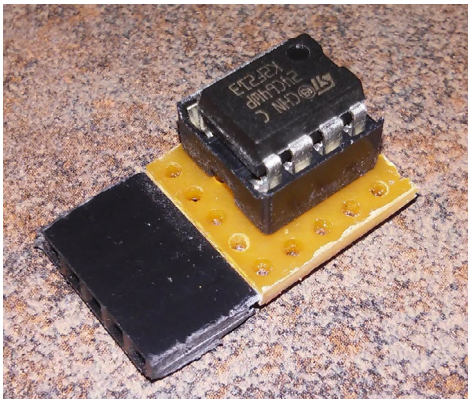
Finally, downloaded my script from http://bit.ly/1mImpB3, save it to the same directory that contains eeprog, and run it:

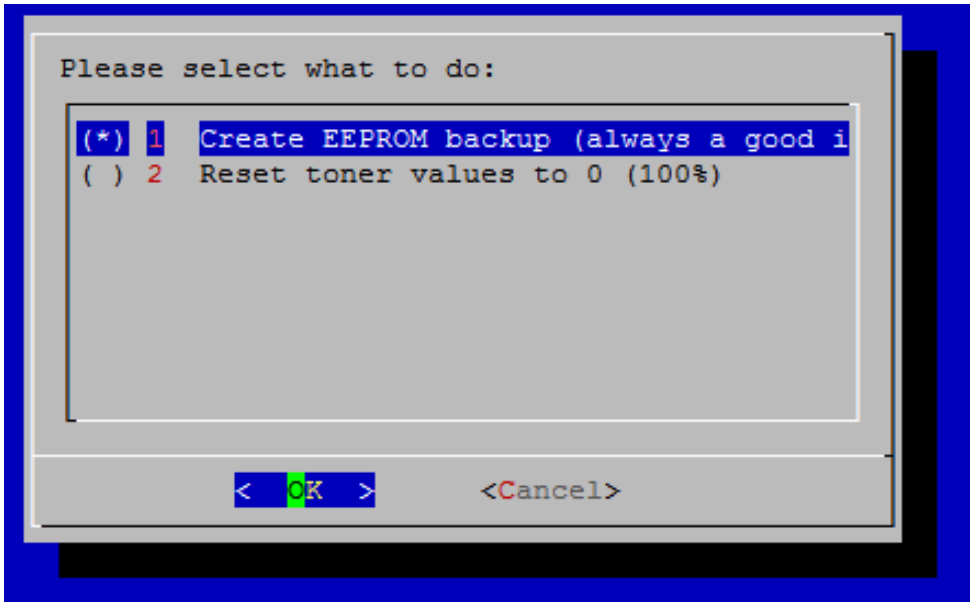
```
$ sudo bash EEPROM_Resetter_v1.sh
```

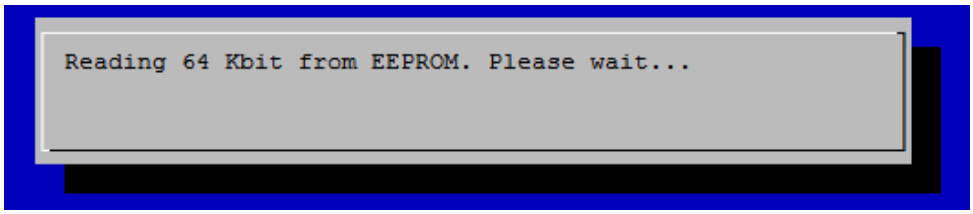
Choose backup and reset, then plug the EEPROM back into the printer again, which allows you to print as much as you want and only refill the cartridges when they are actually empty.

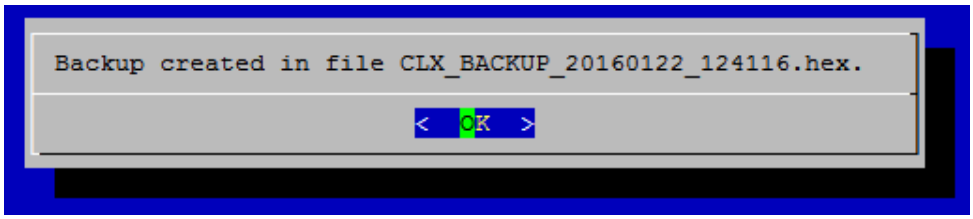
Figures 1 - 3 - The EEPROM connected to the C1

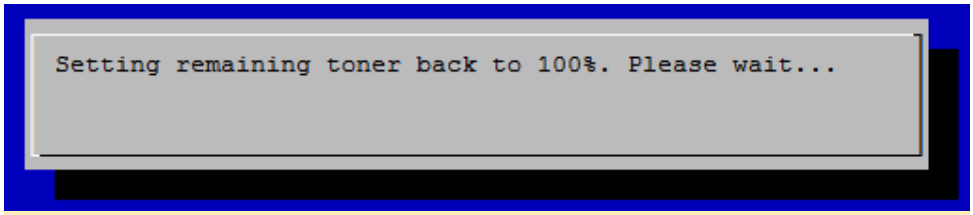












Figures 4 - 8 - Screenshots of the toner reset application

Improvements

The script is not very dynamic, since I only implemented the essential functions for my purposes. In the future, I could implement a parameter for different printers by using different addresses inside the EEPROM, as well as include autodetection of the EEPROM. For comments,and questions, please visit the original post at http://bit.ly/1U8R4F2.

Further reading

Similar project:
http://bit.ly/1QPJidO

Raspberry Pi EEPROM program:
http://bit.ly/1L0mkUF

Hardware design:
http://bit.ly/1RdpzHP

MEET AN ODROIDIAN

CHRISTOPHER DEAN (@TPIMP)
ACCOMPLISHED QT5 DEVELOPER AND HARDWARE VIRTUOSO

edited by Rob Roy

Left to right is Storm, Sheldon, Rogue, Nicole, Christopher, Peyton

Please tell us a little about yourself.

My name is Christopher Dean. I have been working with ODROID development boards for over 4 years. I got my first ODROID for curiosity. After the release of the ODROID-U3, I began using them in school projects. Over the last 8 years, I have studied many aspects of Computer Software, Simulation, and Hardware design. Currently I work and own Protoze, which is a technology research start-up. I am extremely busy and the days are long, but I enjoy the variety of challenges that owning a business brings. I live in the beautiful state of Oregon with my amazing wife Nicole and our four pets, who are like our children: Sheldon, Peyton, Rogue, and Storm.

How did you get started with computers?

Like most people born during the 1980's: video games! My grandpa gave us an old computer with Duke Nukem 2 and other DOS games that I quickly learned how to "execute". Eventually, I moved to Windows 3.1 with a game called "Raptor: Call of the Shadows" and a Sidewinder Joystick. As I got older, I used computers to do school work and edit my skating footage. I have been, and always will be, a PC gamer.

What attracted you to the ODROID platform?

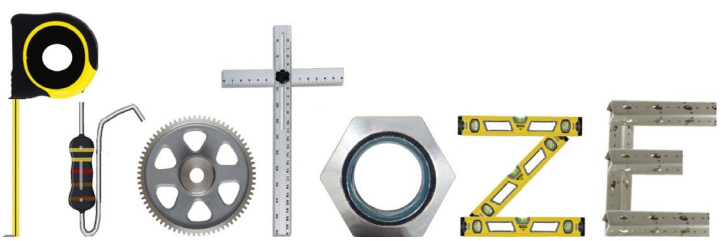
At first, I liked the pure horsepower that the ODROID-U2 provided at its low cost and small size. Over time, I was more attracted to the quality of the boards, combined with the strength of the community. Without the community work and Hardkernel's official Ubuntu images, I would have been in over my head.

How do you use your ODROIDs?

I mostly use my ODROIDs for learning. Over the years, I've learned a great deal about Linux on ARM. Now I use ODROIDs wherever I can! I put them in robots, prototypes, and anything IoT-related that needs USB, large data, or heavy GPU workloads, which is a growing demand. I save a lot of time for my clients by using the ODROID-VU7 and ODROID development boards to turn their product ideas into tangible early builds. Utilizing ODROID and Qt5 (QML) allows me to rapidly prototype ideas in weeks instead of months.

Which ODROID is your favorite?

The ODROID-U2 is my favorite because it was my first



Protoze is Chris' venture, and this is his business logo.

Qt Developer's Dream desktop

one. However, in terms of function, the C1 and C2 are very powerful platforms. The C0 in time will likely become my new favorite, since it provides so much for so little!

Your Qt5 Developer's Dream pre-built OS image is very popular. What do you like about Qt5, and what motivated you to create the image?

Hands down the best thing about Qt is the cross-platform support. In my opinion, C++ is by nature an extremely powerful, cross-platform, fast language. The developers of the Qt Project make a great language even better. I have worked on multiple projects with only Qt as a dependency, and I don't believe there exists a faster platform with which to prototype, and it has far less dependencies than other platforms. My motivations behind the Qt5 Developer's Dream is to help others get the most out of their ODROIDs. I know first hand that being a Qt5 developer is easy and fun, but building a large software platform like Qt5 can be difficult for many beginners. Providing an image with pre-built libraries direct from Qt-Project source allows developers to try the latest Qt, which is 5.6 beta at the moment, on ODROID development boards in three easy steps:

- **Download the Qt5 Developer's Dream image**
- **Flash the image to SD card or eMMC**
- **Code Less, Create More with Qt5**

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

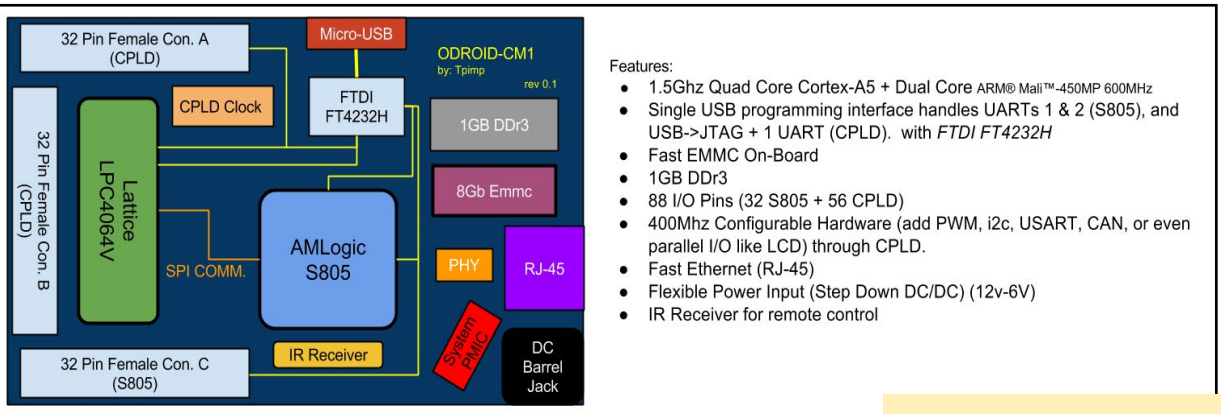
I would like to see more hardware peripheral circuits and IC's to extend the ODROID's ability to interface with more hardware widgets. Each ODROID currently provides an amazing chipset for embedded Linux environments, but I think that the ODROID platform would be even stronger with more hardware peripherals such as SPI, PWM, high speed GPIO. I believe any IoT problem that currently exists can be solved using an ODROID paired with a microcontroller or programmable logic controller. I personally have explored the idea of putting the microcontroller unit (MCU) or field programmable gate array (FPGA) on the same board as the embedded Linux SOC. This is easier said than done, and involves lots of research and development work.

What hobbies and interests do you have apart from computers?

I enjoy skateboarding, video games, open source projects, and spending time with my family.

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

Anyone can write code, but not everyone is cut out to develop software. Whatever you do, don't learn code because you have to. Creating software is like art, because it helps if your heart is truly in it. If you do want to learn, decide what kind of software you want to write. If you want to create an application, a website, a mapping algorithm, or a virus, then do your homework! Learn what language/layer your development will be at, then learn about that language. There are so many resources online and great books written on almost every development language. It's important to understand that learning how to write code is the beginning step to learning how to write bad software. There is still even more to learn before you write "good" software. Be open to criticism, because it can hurt, but ultimately helps you grow.



Chris' dream schematic for the next ODROID