Running Netflix with Linux • ODROID C1+ • Android Gaming • SAMIIO

ODROID Magazine

Year Two
Issue #21
Sep 2015

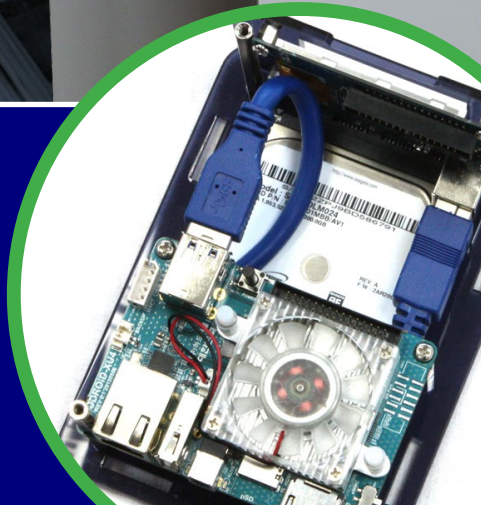
ROBOT Lamp

Learn how to to build a fully automated robotic companion with an ODROID

Logical Volume Basics

Discover a new way to play the piano with Arjuna

Create your own NAS and Cloud with Cloudshell



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

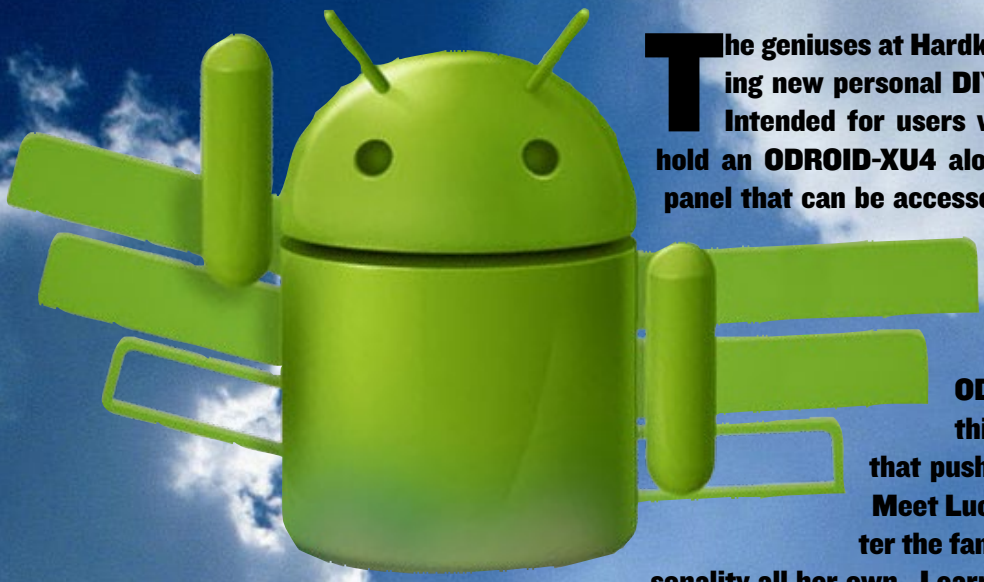
We are now shipping the ODROID-U3 device to EU countries! Come and visit our online store to shop!

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 geniuses at Hardkernel have recently created an exciting new personal DIY cloud server kit called Cloudshell. Intended for users who desire maximum privacy, it can hold an ODROID-XU4 along with a hard drive and 3.2" LCD panel that can be accessed remotely using open source cloud software such as OwnCloud. The ODROID-C1 gets an update with the new C1+, now available for purchase at the Hardkernel store.

ODROID owners are very creative, and this month features several projects that push the boundaries of ARM computing. Meet Luci, the robot lamp, who is modeled after the famous Pixar mascot with a unique personality all her own. Learn how to play the piano using Arjuna, build a single application interface with QT5, create powerful I/O applications with SAMIIO, and manage your disk space better using LVM. We also present two of the most recent community images to be released: Ubuntu Server 14.04 LTS by @meveric, and a unique unified Android/Debian distribution by VolksPC. As usual, we make sure to have fun with ODROIDs by running Netflix under Linux and highlighting two of our favorite Android games: Plague Inc. and Sword of Xolan.

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/IyplmXs.
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

ODROID Magazine

Rob Roy, Chief Editor

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

Robert Cleere, Editor

I am a hardware and software designer currently living in Huntsville, Alabama. While semi-retired from a career in embedded systems design, including more than a decade working on the Space Shuttle program, I remain active with hardware and software product design work as well as dabbling in audio/video production and still artwork. My programming languages of choice are Java, C, and C++, and I have experience with a wide range of embedded Operating Systems. Currently, my primary projects are marine monitoring and control systems, environmental monitoring, and solar power. I am currently working with several ARM Cortex-class processors, but my ODROID-C1 is far and away the most powerful of the bunch!

Bruno Doiche, Senior Art Editor

Bruno managed to laugh at the possibility of having an electrical fire in the heat of editing this same magazine that you are reading due to a very old power cord. What happened? He was remotely setting up his mother's Skype account with the articles open and needed to recharge an iPhone 4 that was close to death, then went straight with the charger to the outlet only to be zapped! Some quick thinking was needed to see what could instantly be unplugged and get everything back to normal. The only casualty was that the power cord melted down.

Nicole Scott, Art Editor

I'm a Digital Strategist and Transmedia Producer specializing in online optimization and inbound marketing strategies, social media directing, and media production for print, web, video, and film. Managing multiple accounts with agencies and filmmakers, from Analytics and Adwords to video editing and DVD authoring. I own an ODROID-U3 which I use to run a sandbox web server, live in the California Bay Area, and enjoy hiking, camping and playing music. Visit my web page at http://www.nicolecscott.com.

James LeFevour, Art Editor

I am 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.

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.

INDEX

CLOUDSHELL FOR ODROID-XU4

A DIY PERSONAL CLOUD SERVER KIT

by Justin Lee

What was the number one request of Linux fans for the ODROID? Well here it is: a case with an awesome USB 3.0 to SATA converter and a LCD console, which also come in white and blue

The Cloudshell for the ODROID-XU4 is an affordable DIY Network Attached Storage (NAS) Solution for creating your own personal cloud! It includes a color LCD console display on the front, a USB3.0 to SATA bridge (Genesis GL3321G), and has room for a 2.5inch HDD/SSD along with the XU4 board mounted inside. It's a great way to have an XU4 with lots of storage inside a portable, compact case.

Specifications

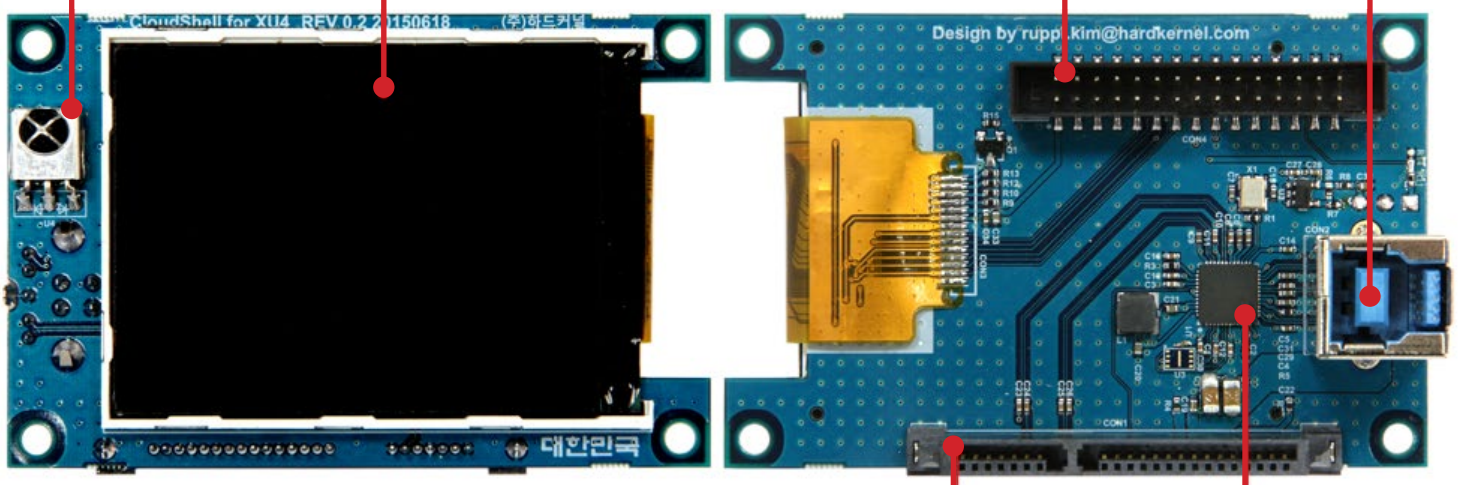
Dimensions: 47mm x 25mm x 21 mm (assembled)
Weight: approximately 248g
Color: Smoky Blue and Smoky White
Circuit board: includes IR receiver, 2.2" 320 x 240 TFT LCD, SATA Connector, USB3.0 Connector and 30-pin I/O Connector
Hard drive bay: 12-14mm (15mm is not compatible)

Further information is available in the Hardkernel wiki at http://bit.ly/1Laq7IS.

More information

Check out a demonstration video of the Cloudshell at `https://youtu.be/0c2CxuFzKtI`. To purchase the XU4 Cloudshell for USD$39 or to obtain step-by-step assembly instructions, please visit the product page at `http://bit.ly/1N3xNm7`.

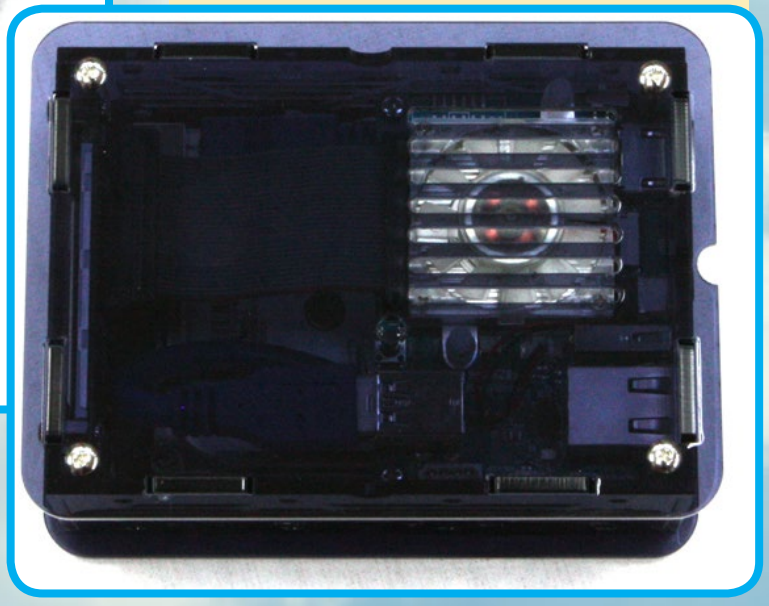
For instructions on setting up a personal cloud server, refer to the January 2015 issue of ODROID Magazine, available for free download at `http://magazine.odroid.com/#201501`



In this annotated diagram, you can see that the Hardkernel team had the genius insight of offering the expansions on a dual sided board containing a LCD, the SATA controller and an extra infrared receiver

When assembling the Cloudshell with hard drive, remember that the bay is compatible with 2.5 SATA disks sized between 12 to 14mm. If you try to assemble it with a 15mm drive it won't fit!

Here is the assembled Cloudshell. Needless to say, it is good practice to keep the ventilation window clear of obstacles in order to allow adequate air flow.



NETFLIX UNDER LINUX ON THE ODROID-C1

CHILL OUT WITH A MOVIE

by @daemon32

Netflix runs well under Android on the ODROID-C1 simply by installing the mobile app from the Google Play Store, but the technique for getting it to work under Linux requires more steps. This article describes how to enjoy Netflix movies while running Arch Linux or Ubuntu.

Installation

Chromium v43.0.2357.134 or higher is needed to run Netflix, which is available for most ARM Linux distributions. You can check the version of Chromium by typing "chromium-browser --version" in a Terminal window. Make sure to upgrade Chromium if the version is lower than the requirement.

1. Download the Chrome OS recovery package at http://bit.ly/1JUOQQw, then extract the .zip file.

2. Install the qemu-nbd package via the package installer available for the operating system that you're using. For example, here is the Ubuntu command for installing qemu-nbd:

```
$ sudo apt-get install qemu-nbd
```

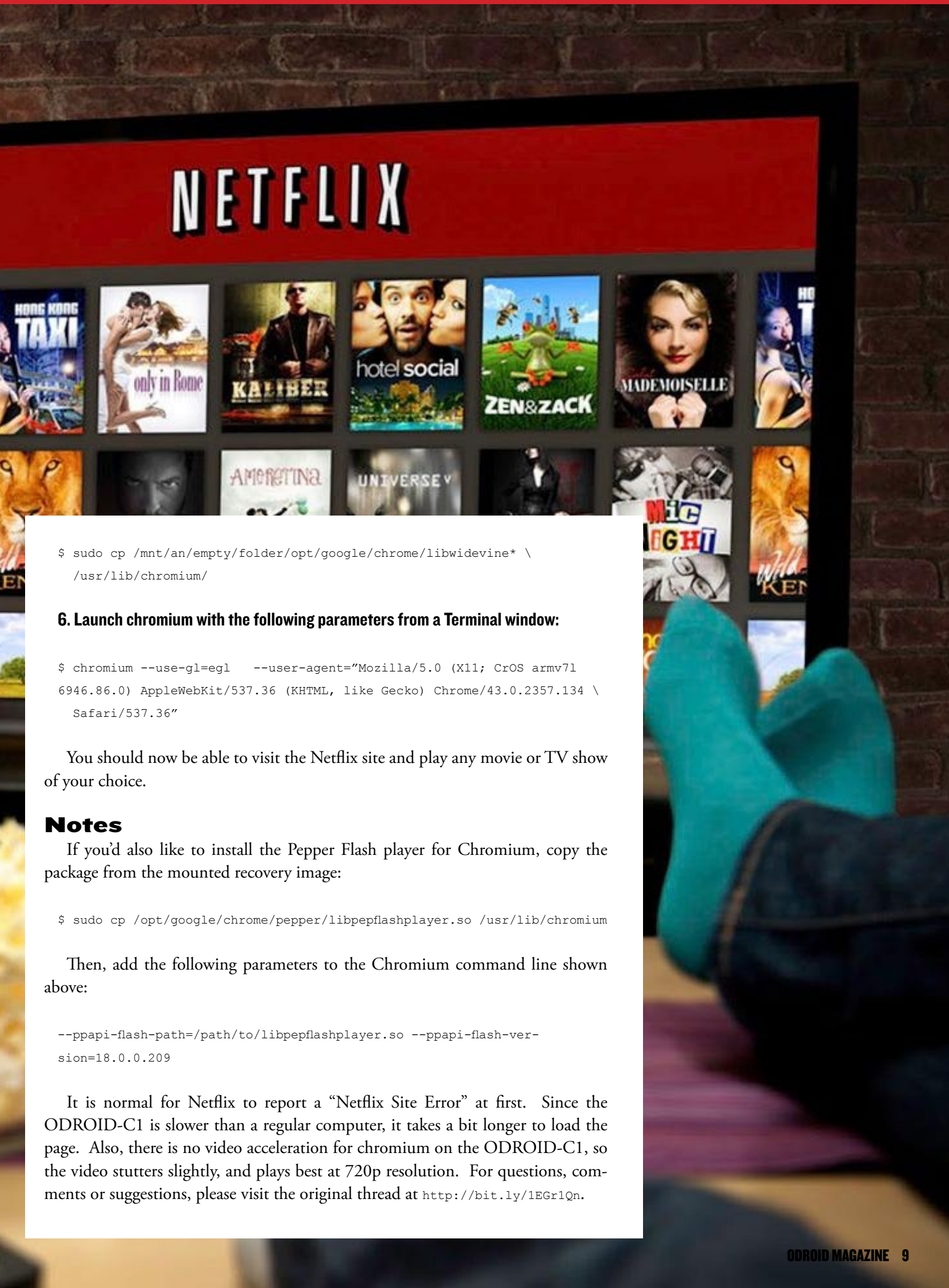
3. Register the NBD protocol:

```
$ sudo modprobe nbd max_part=16
```

4. Mount the recovery image using qemu-nbd:

```
$ sudo qemu-nbd -r -c /dev/nbd0 /path/to/chromeos_6946.86.0_daisy-skate_\
  recovery_stable-channel_skate-mp.bin
$ sudo mount -o ro /dev/nbd0p3 /mnt/an/empty/folder
```

5. Copy the libwidevine files into the Chromium library directory:




```
$ sudo cp /mnt/an/empty/folder/opt/google/chrome/libwidevine* \
  /usr/lib/chromium/
```

6. Launch chromium with the following parameters from a Terminal window:

```
$ chromium --use-gl=egl   --user-agent="Mozilla/5.0 (X11; CrOS armv7l
6946.86.0) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/43.0.2357.134 \
  Safari/537.36"
```

You should now be able to visit the Netflix site and play any movie or TV show of your choice.

Notes

If you'd also like to install the Pepper Flash player for Chromium, copy the package from the mounted recovery image:

```
$ sudo cp /opt/google/chrome/pepper/libpepflashplayer.so /usr/lib/chromium
```

Then, add the following parameters to the Chromium command line shown above:

```
--ppapi-flash-path=/path/to/libpepflashplayer.so --ppapi-flash-ver-
sion=18.0.0.209
```

It is normal for Netflix to report a "Netflix Site Error" at first. Since the ODROID-C1 is slower than a regular computer, it takes a bit longer to load the page. Also, there is no video acceleration for chromium on the ODROID-C1, so the video stutters slightly, and plays best at 720p resolution. For questions, comments or suggestions, please visit the original thread at `http://bit.ly/1EGr1Qn`.

ODROID-C1+

A BOARD FOR EVERYONE

by Justin Lee

The ODROID-C1+ 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 Amlogic 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.

Some of the modern operating systems that run on the ODROID-C1+ are Ubuntu, Android, Fedora, ARCHLinux, Debian, and OpenELEC, with thousands of free open-source software packages available. The ODROID-C1+ is an ARM device, which is the most widely used architecture for mobile devices and embedded 32-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.

Next generation

At Hardkernel, we had received many requests for the following model of ODROID-W. So, we started a survey for components for ODROID-W2. Finding the right CPU was the key part of the project. Our target was a similar cost and performance as the ODROID-W. The Amlogic S805 1.5Ghz quad core processor outperforms the Broadcom BCM2835. We launched the ODROID-C1 in December 2014 and, and the Raspberry Pi 2 was released in February 2015. The ODROID-C1 was superseded by the ODROID-C1+ in August 2015.

Here are some comparisons to give you better understanding of ODROID-C1. Both are Linux-friendly, $35 ARM® single-board computers for various applications and purposes.

Hardware comparison

The ODROID-C1 has many advantages over the Raspberry Pi. The processor is an S805 1.5GHz Quad-core from Amlogic with 1GByte DDR3 RAM, Gigabit Ethernet and infrared receiver. The size of this computer is still only 85 x 56 mm with a weight of 40g, and offers silent operation, 2-3W average power usage, and instant portability, since it fits in a shirt pocket.

One powerful feature of the ODROID-C1 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 40-pin interface header includes SPI, 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-C1, and both arrive with either an Ubuntu or Android 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-C1! Browse the web, play games, run office programs, edit photos, develop software, and watch videos right away. The RTC, IR receiver and ADC features on the ODROID-C1 also offer many options for building great DIY projects.

Specifications

Amlogic ARM® Cortex®-A5(ARMv7) 1.5Ghz quad core CPUs
Mali™-450 MP2 GPU (OpenGL ES 2.0/1.1 enabled for Linux and Android)
1GB DDR3 SDRAM
Gigabit Ethernet
40pin GPIOs + 7pin I2S
eMMC4.5 HS200 Flash Storage slot / UHS-I SDR50 microSD Card slot
USB 2.0 Host x 4, USB OTG x 1 (power + data capable)
Infrared (IR) Receiver
Ubuntu or Android OS

Improvements over the C1

Standard Type-A HDMI connector
Includes heat sink
I2S bus to support HiFi audio add-on boards
CEC function that doesn't require the RTC backup battery
A power path from USB OTG port as well as DC barrel connector
Improved SD card compatibility

Because of the above changes, the original C1 case and heatsink are not compatible. To download software for the C1+, please visit the wiki at `http://bit.ly/1KRKoGV`, and the user manual may be downloaded from `http://goo.gl/iWGYcz`. The C1+ is available for purchase for USD$37 at `http://bit.ly/1Upx5yI`.

Demonstration videos

Hardware introduction:
`https://youtu.be/L1xYBIVBRgk`

Tinkering kit
`https://youtu.be/zocRA1oNY60`

Performance overview
`https://youtu.be/L2ZRW-AagSQ`

LCD shield
`https://youtu.be/SkbZLDl5zTU`

OpenGL ES2.0 with myAHRS+ on Ubuntu
`https://youtu.be/L2ZRW-AagSQ`

Tekken 6 PSP emulation
`https://youtu.be/p8yGS2SHqpA`

UBUNTU SERVER 14.04 LTS

AN OPTIMIZED SERVER ENVIRONMENT FOR YOUR ODROID

by Tobias Schaaf

I am known on the ODROID forums for producing a pre-built desktop image intended for entertainment called ODROID GameStation Turbo. In addition to this image, I also offer a dedicated server image based on Ubuntu 14.04 LTS. In this article, I want to go into detail about this server image and what you can expect from it.

Downloading

Like all of my images, the server image is available on my webspace hosted by Mauro from Hardkernel (@mdrjr) at `http://bit.ly/1N43pXs`, where you can download the image that matches your system. I currently offer images for the ODROID-X, X2, U2/U3, XU3/XU4 and the C1/C1+.

Overview

The image is small and clean, offering a very small server environment with no desktop. It is designed to be a pure server environment. I included my repository on each image, but only for automated kernel updates via apt-get commands. Besides that, it offers a script collection that allows you to easily install and configure server components, which is detailed later in this article.

Differences from official image

My image was originally created from a Linaro server image, therefore the standard login and password is "linaro" instead of "odroid", but since it's very common to rename the default user on a server system and set a new password for that user, this shouldn't be an issue. Besides that, I wanted to increase the security of the system by fixing some issues that are common with the images from HardKernel. For example, if you use the UART to connect to your ODROID on the HardKernel image, you are already logged in as root, which means that no login or password is required and you, or anyone else, has full access to your server and has the power to change, install, or delete whatever they want.

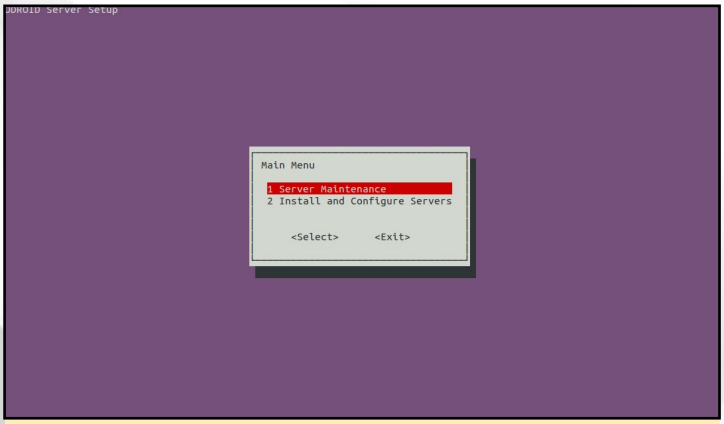
I also deactivated the option to login as root via password to make it harder for attackers to get access to the server, since they normally try to connect as "root" to get full access. However, since this server image has no login for root, this modification creates another layer of security. I also generate new SSH keys on the first boot, so that not every server that is based on my image has the same SSH host key files.

Another issue present in the Hardkernel image is the method of waiting for network timeouts. The default Ubuntu Server image waits nearly two minutes if there's an issue with the network during boot. This means the boot process can be delayed cause the server is idling while waiting for network. Since it's not critical to wait for the LAN, I deactivated that option in order to reduce the boot time significantly.

I also added a script that increases the partition size of the rootfs system to the full size during the first boot. Therefore, the image can be kept very small (around 1GB) which allows for fast flashing, then later resizing of the partition the the full size of the eMMC or SD card on boot. Another major difference is the kernel updates via apt-get that I offer for all ODROID boards, as well as the script collection that I include with my server images.

Script collection

If you are logged into the image, either via SSH, UART or directly with keyboard on your ODROID, you can type "sudo odroid-server", and it will bring up a menu that can

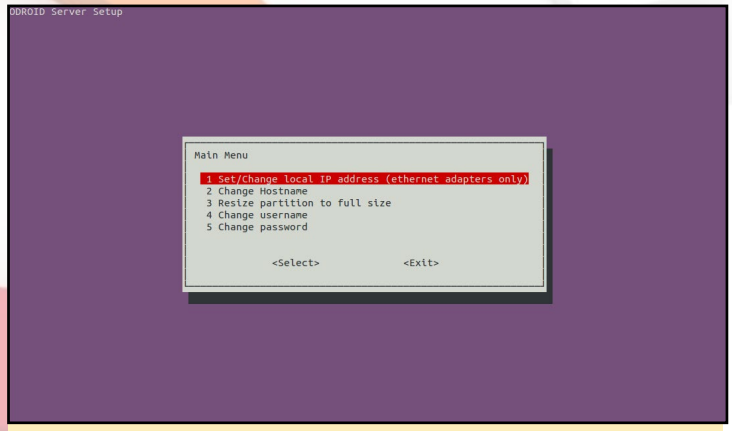


Startup screen of the script collection

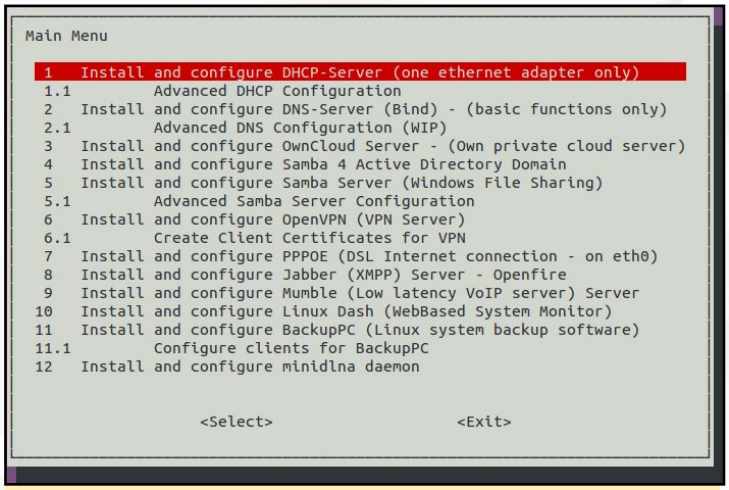
guide you through different server tasks, as shown in Figure 1. From this menu, you can select if you want to configure the server itself or install and configure different services, such as Samba Share, Samba4 Active Directory Domain, OpenVPN server, and many other packages.

I created these scripts in order to make it easier to setup new systems, especially for often used tasks. For instance, it's easy to change the password for users that exist on your system, or even rename a user. As mentioned earlier, the default user "linaro" should be changed to increase security. To do so, go to "Server Maintenance" -> "Change username" and change the user to anything you want. The home folder will be renamed together with the user. The same method applies if you want to change the password for the user.

I tried to make it easy for you to change basic settings like the name of the server, or set up a static IP address for your system, but there is more to it than that. If you want to install a simple Samba share and exchange files over your network, there is an easy way to install a Samba server and add shared folders as well. If you'd like to set up the newest OwnCloud server on your system, it's just a few clicks away with my server scripts.



Maintaining the server itself



Different servers that you can install and manage

These scripts are part of my git repository at `http://bit.ly/1ECywYQ` and can be used on other images as well, but are primarily intended for Ubuntu 14.04 LTS which address some specifics that only exist in Ubuntu 14.04 LTS. Whenever I have time, I try to work and improve the scripts or implement new functions. The scripts are not perfect yet, but I like to fix issues and improve it. This also means other people are free to do pull requests, or ask for new features, or help me figure out some bugs. I will probably restructure the menu over time in order to make it easier to navigate.

Useful tips

• It's best to change the default user and password to something only you know to further increase security on the system.

• These server images are not meant to run Kodi or similar programs, nor to have a desktop to control them. These are pure server images and should be handled as such. Although it's possible to add a desktop and other things, it's not recommended, since there are no libMali.so blobs or framebuffer drivers installed, such as armsoc or Mali DDX.

• Ubuntu 14.04 LTS uses irqbalance, which is normally is used to distribute IRQ interrupts over multiple cores, rather than let them always be handled by the first core. This should improve I/O opts and other things. However, on Ubuntu 14.04, irqbalance has a bug which causes it to use up 100% of available RAM over a period of several days or weeks. Therefore, you should either deactivate or remove this service, or create a cron job that restarts the service once a day.

• It's good practice to add an SSH certificate for the user root for any computer that is allowed to connect directly as root, rather than adding a password for "root". That way, even if the system is hosting services on the Internet, no one can access the server as root with a password, but ths server can still be managed from your local network from a dedicated computer.

UNIFIED ANDROID AND DEBIAN DISTRIBUTION

THE BEST OF BOTH WORLDS

by Vasant Kanchan

Although the Linux kernel has been extremely successful, we are still waiting for a large scale deployment of the Linux desktop. Most of the current desktop deployments are on the x86 platform, and projects such as WINE try to provide a glue layer that allows us to run Windows applications on an x86 Linux desktop. Using Debian on the x86 can reduce the software licensing cost, but a move to an ARM based Linux PC can also reduce the hardware cost as there many semiconductor manufacturers selling cheap and reasonably powerful ARM SOC's. Debian desktop software running on an ARM based PC could provide a low cost computing solution for the underdeveloped world. In fact, many countries use Android tablets in their colleges as a part of their e-learning program. Android has become extremely successful, and fortunately is based on the same Linux kernel used by the Linux desktop. Independent software developers are also interested in writing applications for Android, which has been a big problem for Linux desktops because of their limited penetration

VOLKSPC

At VOLKSPC (http://www.volkspc.org) we have developed a solution to integrate both Android and Debian with the following features:

• Full support for running Android applications.
• Multi-windowed Debian applications that work well with the keyboard.
• Instantaneous switching between the Android and Debian desktops.
• Quick launch of Android applications from the Debian desktop.
• Users can install applications from both Google play and the Debian repository.

This unified distribution, based on Android KitKat and Debian Jessie ARMHF, currently runs on the ODROID-C1. Even though I frequently refer to the Debian desktop in this document, this solution is applicable to other X-Windows based distributions such as Fedora and Ubuntu.

Benefits of running Android With Debian

Android has many advantages such as:
• Large number of available applications and games.
• Very good support for touch interface.
• Hardware accelerated multi-media.
• Large number of ARM SOC's with Android implementation.
• Runs on enhanced Linux kernel.

However there are some Android limitations with used with large displays:
• Each application uses the whole screen.
• Cannot view multiple applications at the same time and cannot easily switch between them.
• No support for desktop style word processing, email, etc with keyboard and mouse input.

Linux desktop distributions such as Debian have very good support for legacy desktop-style applications such as LibreOffice, the Firefox browser, and the Thunderbird email client. These applications also work very well with keyboard and mouse input.

A unified Android and Debian distribution would be a good solution for some classes of devices or users:
• A large screen tablet with detachable keyboard such as the Asus Transformer. For this class of devices Windows 10 is too expensive and Android is too limited.
• A solid-state cloud PC with attached keyboard and mouse.
• In underdeveloped economies, consumers may not have access to multiple computing platforms. A single device should meet all their computing needs. Android by itself cannot fulfill this need.

Implementation Overview

Android and Debian desktop share the same kernel, so despite the fact that the user space libraries are different, both applications can run simultaneously. The biggest difficulty in integrating the two systems is with the graphics technology. Android uses SurfaceFlinger and Debian uses X-Windows for drawing graphics.

Unified Android and Debian with VNC

A common approach to running Debian on Android is to send X11 graphics to a VNC viewer running on android (ref: http://whiteboard.ping.se/Android/Debian).

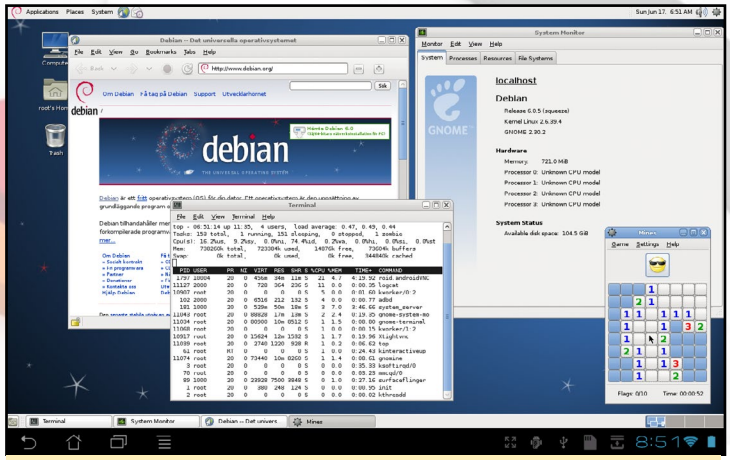


Figure 1 - Android-Debian

The Debian distribution with its own set of libraries can run in a chroot and not interfere with any Android libraries. With VNC, Debian applications can redirect all graphics to Android. There are two problems with this approach:

- **Graphics commands are sent through several layers of software, resulting in poor performance.**
- **Some screen real estate is lost to the Android panel.**

Although this is a good proof of concept, it is not an acceptable solution.

X Window System

The X Window System is a network-oriented windowing system used by all Linux desktop distributions. The X server controls the display ,and is responsible for the drawing of on-screen graphics. Typical applications are X clients which exchange messages with the X server via the X Window protocol. Applications are typically written through high level toolkits such as GTK, QT, Tcl/TK, etc.

The Figure 2 shows an application communicating with the X server in a Linux desktop environment. Most of the complexity of X Windows is due to the client-server architecture. The design aspects that make X Windows complex and slow are:

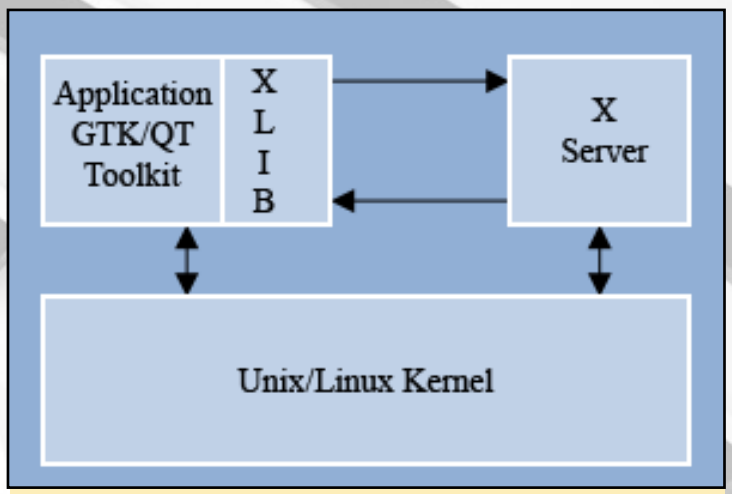


Figure 2 - X Windows

- **Both the client and server have to buffer and format commands and responses as per the X Window protocol.**
- **Synchronous and round-trip requests are inefficient.**
- **Frequent context switching between applications and the X server degrade performance.**
- **Graphics rendering can only start after the X server starts running.**

There are efforts within the open source community to move to new display server technologies such as Wayland and MIR.

MicroXwin Graphics

MicroXwin implements graphics processing in the kernel as a load module and provides a character driver interface to the associated X11 library.

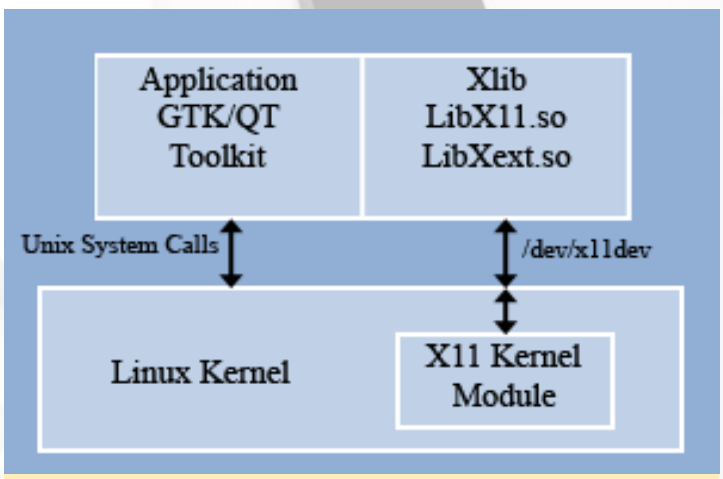


Figure 3 - MicroXWin

The advantages of this design are:

- **Low latency and round trip.**
- **Minimal buffering of requests and responses.**
- **No context switch overhead.**

NEED SOMETHING TO DO WHILE YOU'RE SICK?

EXTERMINATE MANKIND WHILE YOU RECOVER WITH PLAGUE INC.

by Bruno Doiche

The worst thing about catching a cold or flu is when you graduate from "well, a couple of days rest; I'll enjoy the free time!" to: "Netflix binged, Internet fully browsed, Odroid forum totally read, now what?" So, if you have ever wondered, what would if be like if you are patient zero of a plague that is bound to end all humans? Now you can do your best to be the worst thing ever to happen to human civilization since the black plague in 1348!

https://play.google.com/store/apps/details?id=com.miniclip.plagueinc&hl=en

Now do you see what happens when you don't wash your hands after using the bathroom?

- **Direct rendering of all graphics by the client.**
- **No changes required for existing Xlib applications.**
- **2X faster than standard X-Windows.**

Unified Android and Debian with MicroXwin

With MicroXwin, Debian applications can write directly to the frame buffer without going through Android's SurfaceFlinger. This requires that both MicroXwin and Android graphics co-exist and run simultaneously without interfering with each other. In addition to providing Xlib API support to Debian applications, the MicroXwin kernel module also provides following features:

- Monitors keyboard shortcuts (LeftALt + LeftMeta) and facilitates switching between the Android and Debian displays.

- The state of the Android graphics and Debian graphics is maintained.

- At any given instance, only Debian or Android occupies the whole screen, and switching is instantaneous..

- Applications are unaware of which desktop is currently being displayed on the screen.

- Provide a simple API for switching between the Android and Debian desktop.

Application performance with a unified distribution will be similar to what is expected when they run under their respective environments. However since many more applications will be running after boot up, it will be necessary to provide sufficient system memory.

Performance of the unified distribution on the ODROID-C1

This unified distribution https://www.youtube.com/watch?v=USNISy17-YU has been ported to the ODROID-C1 board and a download link is available on the ODROID-C1 general forum. We have ported the XFCE desktop that is available on Jessie.

We used gtkperf to measure graphics performance on both Ubuntu and our unified distribution.

On Ubuntu, gtkperf took about 43 seconds to complete versus 12.62 second on our unified distribution - a 3X speedup.

The Debian lxtask application reports 566MB of free memory at a display resolution of 720P, and about 500MB of free memory at 1080P. So there is a slight increase in memory usage. If necessary, the user can exit completely from the Debian XFCE desktop.

XU4 CLUSTER

A THOROUGH LOOK AT SEVERAL AVAILABLE OPTIONS FOR HIGH PERFORMANCE COMPUTING

by Alan Mikhak

A XU4 cluster being managed by an XU3 console

I evaluated a set of ODROID-XU4/XU3 devices as an embedded cluster platform for parallel computing with OpenCL and MPI using Heterogeneous Multi-Processing (HMP) Linux. As a software consultant, the purpose of this evaluation was to add more tools to my toolbox so that I can offer better service to my clients.

Project goals

I started with a few requirements in mind. I was looking for a platform based on ARM Cortex-A9 CPU cores at a minimum, paired with a GPU which could run Ubuntu Unity desktop with hardware accelerated graphics, supporting OpenGL, OpenCL and CUDA with floating point and vector math hardware. I just didn't want to run the cluster nodes in headless mode. Having a popular desktop on each node was and still is on my wish list. A WiFi interface would also be a bonus so that I can evaluate wireless clusters.

Evaluations

At first, I considered platforms based on the Freescale i.MX6 Quad processor, which has four Cortex-A9 cores. I evaluated the Wandboard Quad board which pairs a Vivante GC2000 GPU with the i.MX6 CPU. It also includes a Broadcom BCM4329 WiFi interface. I had used a Wandboard Duo a few years before with an earlier version of Ubuntu desktop running on it. However, no matter what I tried, I just couldn't get Ubuntu 14.04 Unity desktop to run on the Wandboard Quad and detect my Samsung SyncMaster BX2431 HDMI monitor at the time that I tried.

To run Ubuntu 14.04 Unity desktop smoothly, I would need a platform that supported the full desktop OpenGL, not just OpenGLES. I also needed hardware acceleration for X-Windows. I started considering platforms based on Samsung Exynos 5 Octa which has four Cortex-A15 and four Cortex-A7 CPU cores as well as an integrated ARM Mali T628MP6 GPU. I purchased a Samsung Chromebook II which ran ChromeOS with graphics acceleration. The Chromebook has a built in monitor, keyboard, track-pad, and WiFi. This seemed to be an advantage in my case, considering the cost of the USB HDMI KVM switches and HDMI monitors. Each Chromebook II node would be self-sufficient. I put the Chromebook II in developer mode and installed Ubuntu 14.04 but again couldn't get Unity desktop to run on it at that time.

I then evaluated an Arndale Octa board which is also based on the Exynos 5 Octa. Arndale Octa seemed promising because www.linaro.org listed recent support for it. Again, no matter what I tried, I still couldn't get Ubuntu 14.04 Unity desktop to come up on the Arndale Octa board and detect my Samsung HDMI monitor.

I moved on to evaluating the ODROID-XU3 which is also based on the Exynos 5 Octa. The Hardkernel website showed that the Ubuntu MATE desktop was already running with smooth acceleration. HMP support was also on the horizon in Ubuntu Linux kernel. The ODROID-XU3 was the best match so far because it actually booted into an Ubuntu graphical desktop with my Samsung HDMI monitor. However, I decided to keep looking for a platform that could run the Unity desktop which I was used to.

I switched to evaluating the Nvidia Jetson TK1 development kit based on the Nvidia Tegra K1 SoC which integrates four ARM Cortex-A15 CPU cores and 192 Nvidia Kepler GK20A CUDA GPU cores. The Jetson TK1 is actually the first platform I evaluated that runs Ubuntu 14.04 Unity desktop smoothly with graphics acceleration. I then added a second Jetson TK1 to bring up a two-node cluster for evaluating

SLAY THE DRAGON, SAVE THE VILLAGE

SWORD OF XOLAN PROVES THAT NO MATTER HOW HIGH DEF OUR DISPLAYS GET, WE WILL ALWAYS LOVE PIXELS

by Bruno Doiche

One of my all time favourite games ever was an arcade game called Black Tiger, a classic Capcom platformer that blended adventure with some RPG elements, where you would look for coins in order to get better items and progress in the game. I love to emulate it whenever I can, and couldn't be happier to stumble across this gem from a Turkish indie game developer that managed to capture the thrill that I felt when first playing it. It comes with the added bonus of having excellent controller support!

https://play.google.com/store/apps/details?id=com.Alper.SwordOfXolan&hl=en

Platforming, check. Runs on all ODROIDs, check. Controller support, check. Awesome swashbuckling fun, check!

CUDA-aware MPI programs. The only missing requirement was OpenCL. The Jetson TK1 doesn't support OpenCL even though the Nvidia Tegra K1 SoC does.

To evaluate both CUDA and OpenCL on an MPI cluster, I would need to get into custom boards based on the Tegra K1 SoC. Alternatively, I could bring up a second cluster for evaluating OpenCL with MPI and leave the evaluation of CUDA and MPI to the Jetson TK1 cluster. Right about that time, I saw the news announcing the ODROID-XU4 at about half the price of the ODROID-XU3, Jetson TK1, and Arndale Octa. That's how I ended up with a five-node cluster of four ODROID-XU4s and one ODROID-XU3, which is working well so far.

Final setup

Bringing up the cluster of four ODROID-XU4 boards was fairly straight forward. I ordered the boards from www.ameriDroid.com and some aluminum standoffs from www.pololu.com. Both vendors delivered the components extremely fast, and everything worked right out of the box. I didn't have to return any boards at all, which was a relief and a timesaver. I stacked the boards with 1.25" 4-40 thread M-F standoffs, using 0.25" standoffs for the bottom of stack, and fastening the stack with 5/16" screws.

The XU4 boards share two ASUS VX238H HDMI monitors and two pairs of USB keyboards and mice, alongside one ODROID-XU3 and a cluster of Nvidia Jetson TK1 boards through two Bytecc 4-port USB HDMI KVM switches, which came with USB+HDMI cables and convenient infrared remote controls. I purchased a powered USB hub in order to supply power to one of the keyboards, a Corsair K70 gaming keyboard with lights, without which my Microsoft mouse wouldn't work when connected to XU4 USB 2.0 port through a USB switch. I initially tried using Bytecc 5-Port HDMI switches for sharing the monitors while using separate USB switches for the mice and keyboards. The caveat was that neither ASUS VX238H nor Samsung SyncMaster BX2431 HDMI monitors were detected correctly. So, I ditched that switching scheme in favor of using Bytecc 4-port USB HDMI KVM switches.

I flashed the Ubuntu 15.04 image for XU3/XU4 boards onto four 16GB Patriot microSD UHS-1 cards which I inserted into microSD slot on XU4 boards. With the boot mode switch set to boot from microSD, three boards came up to Ubuntu MATE desktop. The fourth board also booted properly after I replaced its Patriot microSD card with a fifth card. I confirmed that all 8 cores were in use by Ubuntu in HMP mode.

Fans often turn on and off randomly, generating some noise during operation. After shutting down Ubuntu and turning the A/C power off, the red LED stays on, although unplugging the HDMI cable turns the red LED off. The cluster is connected through a NETGEAR GS-116 16-port Gigabit Ethernet switch. I used a simple MPI program with MPICH to verify operation of XU4 and XU3 as a cluster. Everything is working well so far.

LUCI, MY LAMP

A SUPERCOOL ROBOTIC COMPANION POWERED BY AN ODROID-U3

by Jochen Alt

A lamp is nearly useless in the daytime. By itself, it does not talk, cannot dance, and is boring. This is where I try to help with my new project called Luci. Luci is an autonomous lamp with a webcam, a high power LED in the lampshade, and five servo motors. She is controlled by an ODROID-U3. Once switched on, she looks around, checking the environment for any sentient beings, and then does what she wants. Check out her video at http://bit.ly/1hLVtOt.

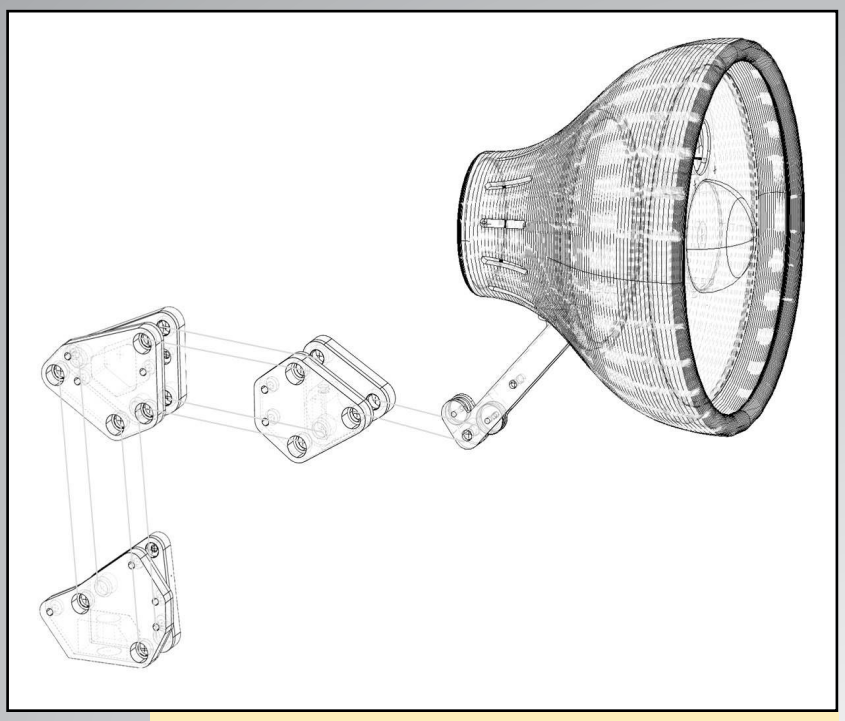
Mechanics

The riskiest part of the projects is the mechanics, so I started with that. I bought a desk lamp, measured the basic dimensions and tried to give it more childlike characteristic by making the lampshade too big and the arms too short. TurboCAD helped to model the lampshade and the hinges.

After having spent many hours in the basement and cutting the hinges out of multi-layered birch wood, it turned out that the friction between arm and joint was too high. So, I reconstructed it with ball bearings, went back to the basement, tried again, and noticed that the springs needed different mounting points, changed that, went back into the darkness and breathed in the fine dust again. Especially after adding the ball bearings I was glad to have used TurboCAD, since this required four

slices per hinge with very precise drill holes.

I did not want to repeat that dusty experience with the lampshade, so 3D printing became very tempting, and I wanted to try it out anyway. I can't report too much on that, since I just gave the CAD model to a 3D printer shop. The biggest challenge of that step was to withstand their desperate attempts to sell me a printer.

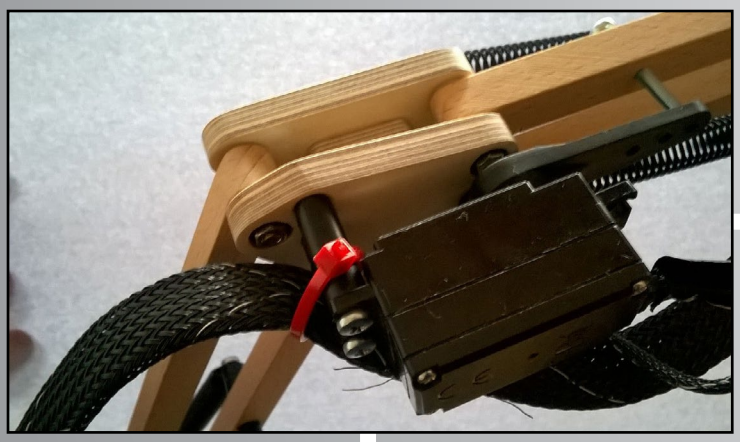


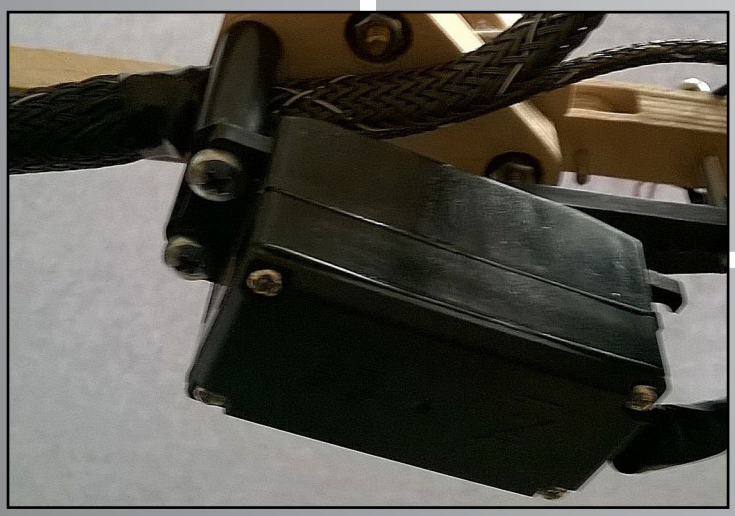
CAD diagram of Luci

Hardware

I started with a BeagleBoard, but it became clear soon that it is too slow for facial recognition. So I ordered an ODROID-U3. Unfortunately, it does not provide five PWM outputs for the servos, so I needed a separate µC to produce the PWM signal for the servos and the High Power LED. In addition, the ODROID is sensitive to voltage flaws, so after having connected the servos to a separate board with its own power supply, sudden resets of the ODROID did not occur anymore. The ODROID with the self-made Arduino shield was supposed to be placed in the base housing of the lamp but, in the end, I did not place it there, since I was wary of having a hot ODROID under full load within a wooden box full of splints.

Hinge closeups





Software

The program runs under Ubuntu using C++ with multiple threads in order to leverage most of the four cores. The first thread grabs the image from the webcam; a second thread takes these images and tries to find all faces currently looking at Luci. The result is a list of faces, and Luci focuses on the biggest one.

A third thread runs the trajectory planning algorithm, which produces a sequence of points and orientations in 3-dimensional space generated by certain patterns. When no face is detected, Luci runs a search pattern looking for faces by sampling the environment until a face has been found. Then, Luci carries out a pattern simulating real emotions like nodding without knowing why, pretending to listen, coming closer or retreating depending on the movements of the face. It's like in real life at home.

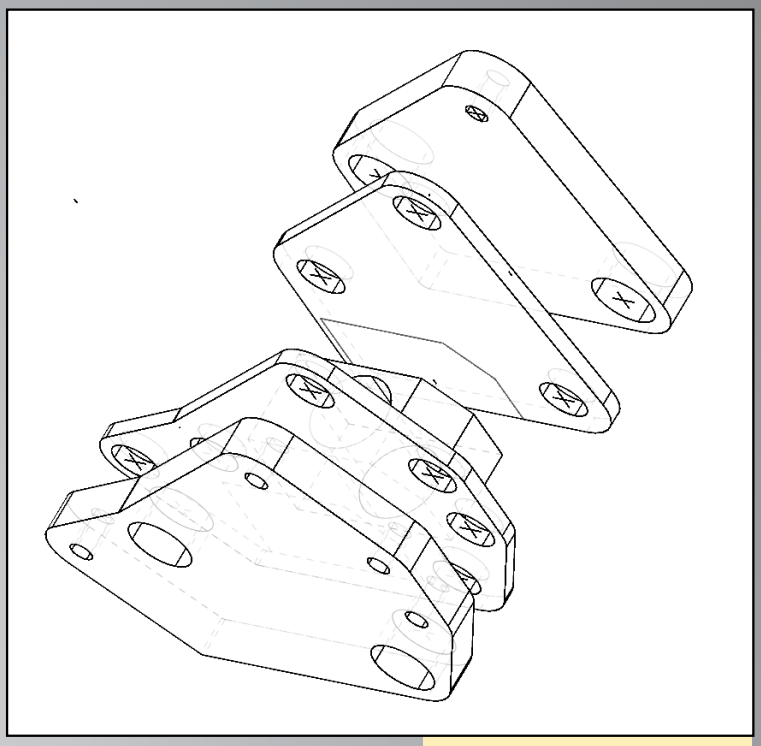
Trajectory Planning

The implementation of the trajectory patterns is rather simple: whenever Luci runs short of queued points to be approached, she invokes a pattern point generator which is parameterized with the current pattern. There, the next point of a predefined sequence of movements is generated. In case of the pattern that interacts with a face, this is:

Move back quickly (surprise, recognizing a familiar face)
Move slowly towards the face (first shy contact)
Watch face from various angles (closer inspection)
Move down and watch the face from a frog's perspective (looking cute)
Go up again and nod (pretending an agreement)

The ideas were borrowed from Eliza (http://bit.ly/1Co34ab), but coded in a robot instead of Emacs. Some patterns with special movements are hard-coded. For example, when Luci pushes a box from the table or looks guilty for watching dirty pictures (1:34 and 2:00 in the video).

Finally, the main loop takes the previous and next point of the trajectory and interpolates all intermediate points with 60Hz using a cubic Bézier curve to smooth the movement. The support points of the Bézier curve are geometrically derived from the trajectory's prevprev (A) and nextnext (D) point by the rule shown in the picture: Since any polynomial with higher grade tends to oscillate when support points are too far away, I kept them in a constant distance of |BC|/3 to B resp. C.

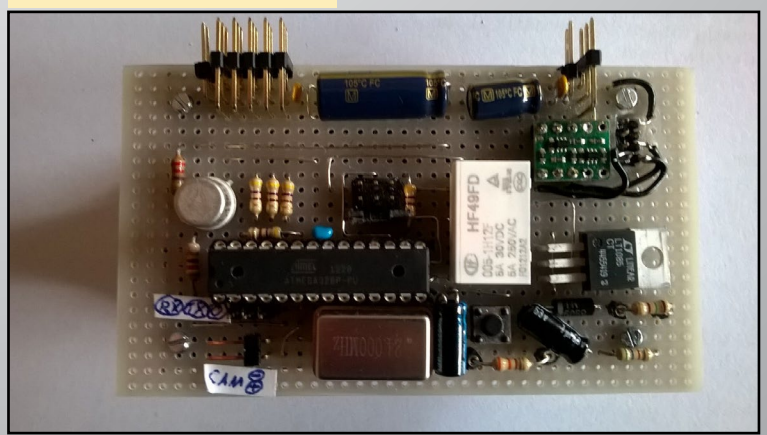


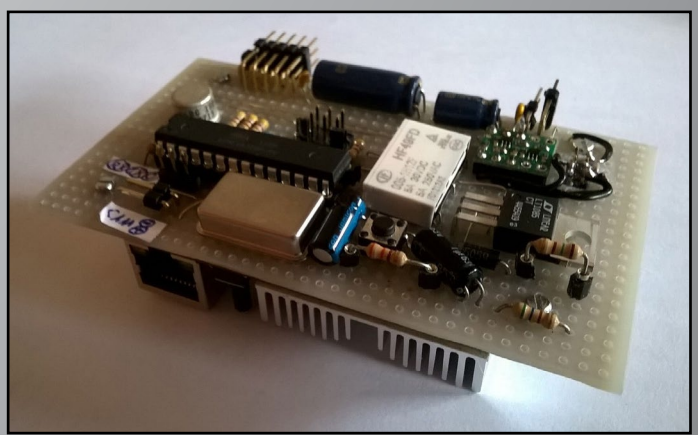
Hinge explosion diagram

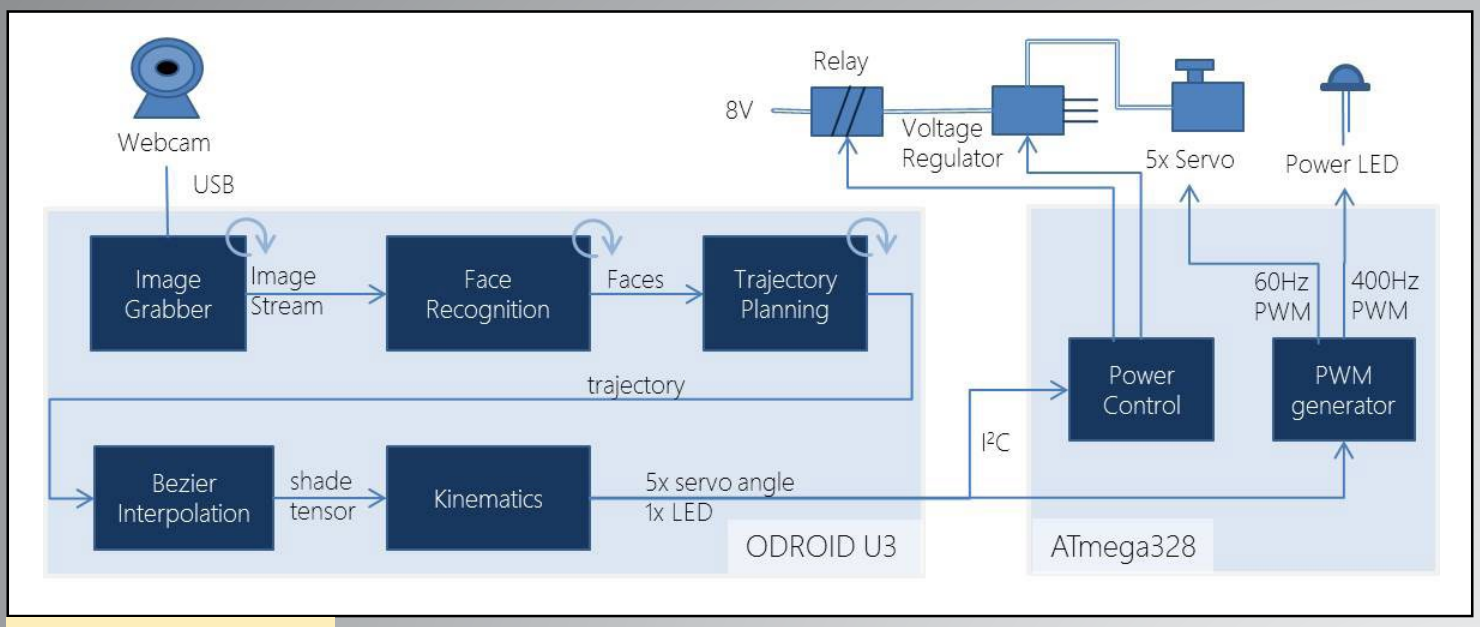
Mass inertia

The last step also computes the lampshade's acceleration, since the Bézier curve does not take into account that, in total, 400 grams are moved. As a consequence, I limited the mass acceleration by ½ g to prevent flapping caused by the elastic construction and the backlash of the servo motors. This is done by checking whether the next position can be reached without accelerating above the limit. If not, the new position is computed by taking the current position and adding the maximum dis-

PCB closeups







Software structure

tance (on the basis of the current speed and maximum acceleration capped by ½ g) along the current speed vector. In the end, the result curve leaves the Bézier curve where it is too sharp. Professional robots do this on the basis of fully modelled inertia, but at this point the methematics tired me out, so I did not try that.

Kinematics

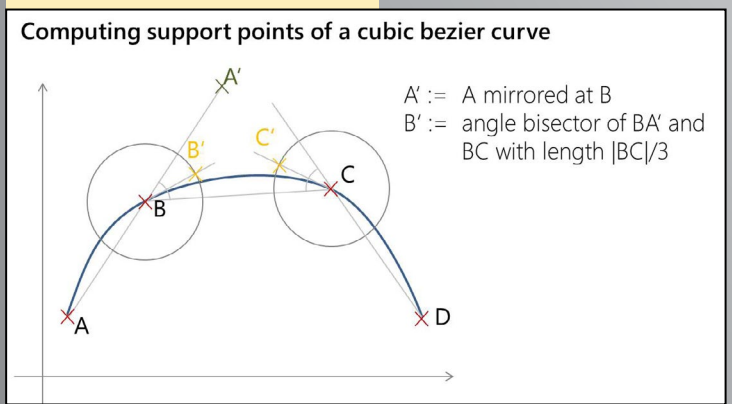
The output of all this is a 3D point which is passed to the kinematics module that computes the angles of all servo motors. This part is textbook robotics, it works as follows:

The algorithm starts with the point / orientation (=tensor) of the head's centre A. First step is to compute position B and C out of the head orientation. This can be done by computing the point C relative to the position of A (C.point-A.point), rotating that by the orientation of the head (A.rotation), and adding it to the point A:

```
C := A.point + rotate(C.point-A.point, A.rotation)
```

Then, the base angle at F, which is the servo angle, can be computed by:

Bezier support points diagram




```
F.angle := atan2(A.point.z, A.point.x)
```

The angles at E and D are computed by considering the triangle EDC and computing its angles with the cosine law:

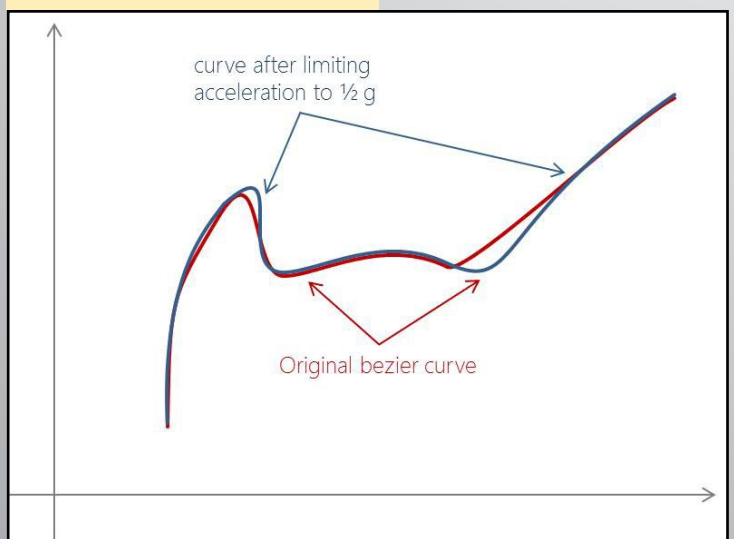
```
E.angle := 90° + acos( distance(E,D)2 +
  distance(E,C) 2 - distance(D,C)) /
  (2*distance(E,D) * distance(E,C) )
```

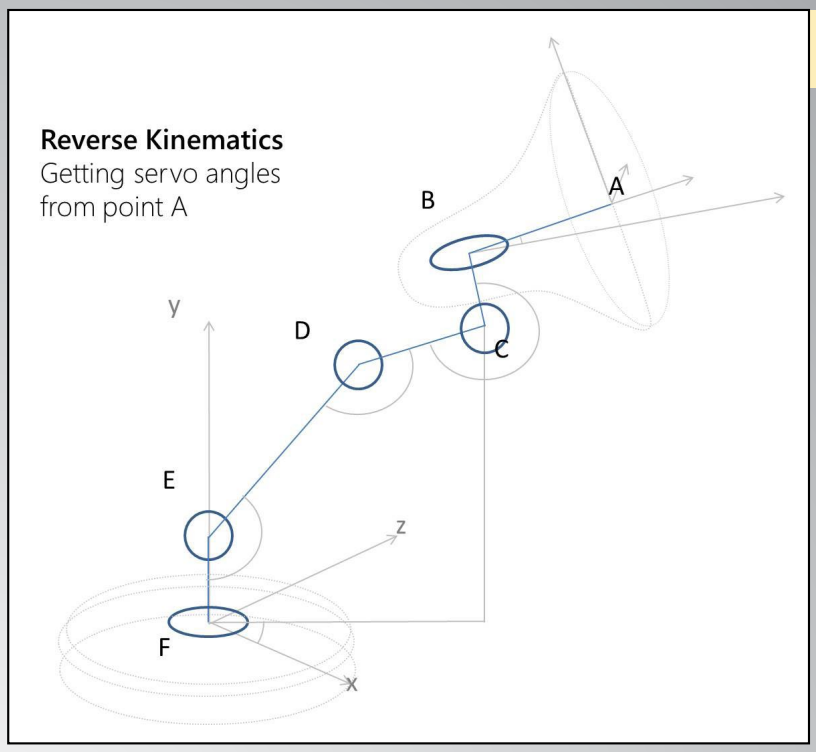
The angle at D is computed in the same manner

```
D.angle := acos( distance(E,D)2 +
  distance(D,C) 2 - distance(E,C)) /
  (2*distance(E,D) * distance(D,C) )
```

The ,ast servo is C, which angle is the orientation of the head around the z-axis minus the angle of CD to the horizon:

Acceleration limit diagram





Kinematics diagram

```
C.angle := A.rotation.z  + 270° -
  acos( C.point.y-E.point.y  /  C.distance(E) )
```

These angles are passed via I2C to the ATmega, where the Arduino library generates a 60Hz PWM signal for the servos. In the beginning, I was concerned about the high CPU use of 3D kinematics, and tried to implement it with fixed point integers and interpolated trigonometry, since I was used to a 24MHz ATmega. What a surprise it was when I recognized that using floats and sin/cos with no caching or table lookup had no noticeable performance impact.

Facial recognition

The facial recognition module uses OpenCV 3.0 with Haar cascade classifiers. Although the newer LBP cascades are significantly faster, they had many more false positives, so I thought 10 fps with Haar cascades is sufficient. From the 2D position of the detected face, the 3D position is estimated assuming a standard face size, which worked surprisingly well. Later on, Luci's trajectory planning module moves towards the face if it is very close, in order to simulate real interest, then moves away if it violates the European intimacy distance. Tracking a face in real time was a bit tricky, since grabbing images from the video stream and facial recognition has a latency of 250ms . So, the computation of the face's 3D position needs to be done relatively to the webcam's position 250ms ago. Consequently, when Luci moves quickly, this does not work well when the image becomes blurry, so the orientation of the head is directed towards the last stable face position until Luci moves slower and following the face in real time becomes possible again.

The trajectory planning module computes the next two points in advance for calculating the Bézier curve. Therefore, the detected face position is not valid anymore when the Kinematics module is sending a position to the servos a couple of seconds afterwards. The solution is to permanently compute the current 3D position and send that to the Kinematics module, in order to change the head orientation towards the face in real-time.

Conclusions

Similar to my experience with my work at my job, everything just took longer than expected. Surprisingly, the software and hardware worked out quickly, but getting the construction and the mechanics in a shape that worked without being too heavy, with properly mounted servos and springs, took me a couple of weekends in the basement. The mathematics were definitely challenging. Getting facial recognition done was the simplest part, but gets the most ahhs and ohhs. The guys from OpenCV did a pretty good job at making this really easy. The most fun part was the trajectory planning, such as how Luci should move when the webcam recognizes a face moving.

I have been thinking of enhancing Luci with a microphone and a beat detection module allowing her doing cool moves according to the music. My first thought was that it wouldn't be difficult, but rhythm detection seems to be extremely tricky.

Parts List

ODROID-U3 running Ubuntu 14.04.02 with the latest g++ compiler

Software uses OpenCV 3.0 and Boost 1.57 as base libraries

ATmega 328 running C++ firmware based on Arduino Library for Servos and I2C

Servos from Hitec: 77B (for turning base & nicking the head), 7954SH (lower strut, strong & expensive), 7775MG (upper strut, also expensive), 5065MG (turn head inside the lampshade)

3D print of a TurboCAD model made of ABS

Springs from my local dealer, 20 ball bearings, 0.5m2 multi-layered birch, and several brass axis

Source code on http://bit.ly/1iq9amT

Other projects

Two years ago, I was bored by using Excel too much at the office, since I'm a professional software architect, so I started learning electronics and embedded technology and made my first robot, which can be viewed at http://bit.ly/1VHg7OX. His name is Paul, and it balances on a ball.

ARJUNA

AN ODROID-BASED PIANO TEACHING DEVICE

by Ilham Imaduddin

Many people would love to learn how to play the piano, but becoming proficient on this 88 key instrument is quite difficult. Many people have a problem with recognizing and understanding the scale of a note relative to the key's position. Another problem occurs when the player attempts to balance the fingers of the right and the left hand while playing different notes. In view of these common learning problems, we had an idea to create a device that would help piano students with learning some of the basic skills of piano playing in a simple fashion.

In this article, we are introducing 'Arjuna', a device that will assist the student by guiding each finger, of both hands, to the correct note in the key which is being played.

Arjuna Components

Arjuna consists of two primary components, the MPU (MIDI Processing Unit), and the Hand Module.

Here are the components for both parts:

MIDI Processing Unit
- ODROID-C1
- nRF24L01+ Transceiver Module
- Keypad

Hand Module
- Arduino Pro Micro
- nRF24L01+ Transceiver Module
- Small Vibration Motors
- 1 Cell LiPo Battery
- 5V Step-up Regulator
- 3D Printed Ring
- 3D Printed Case

Students can use this device with either a traditional piano or a digital piano.

Arjuna Function

The function of Arjuna is straightforward: the MPU receives data from the instrument, processes that data, and issues a command to the Hand Modules to guide the student. The MPU and MIDI keyboard are connected to either the keyboard's USB cable, or to the keyboard's MIDI port by using a MIDI-to-USB Converter. Therefore all keyboards and digital pianos with a USB-MIDI or a standard MIDI port are compatible with this device.

For the MPU, we choose the ODROID-C1 because of its flexibility, power, and ease of use. A transceiver module based on the Nordic nRF24L01+ IC is used to wirelessly connect the MPU to the Hand Modules. The transceiver is connected to the ODROID-C1 through the SPI interface. MPU control is performed by using a keypad, which is interfaced to the ODROID-C1 through its General Purpose Input / Output (GPIO). The code for the SPI and GPIO I/O was written to use the WiringPi library.

Arjuna Hand Module

The Hand Module is a collection of five 3D printed rings, one for each finger, with two vibration motors on each ring. The vibration motors are placed on both the left and right side of each ring. This is used for guiding the student to

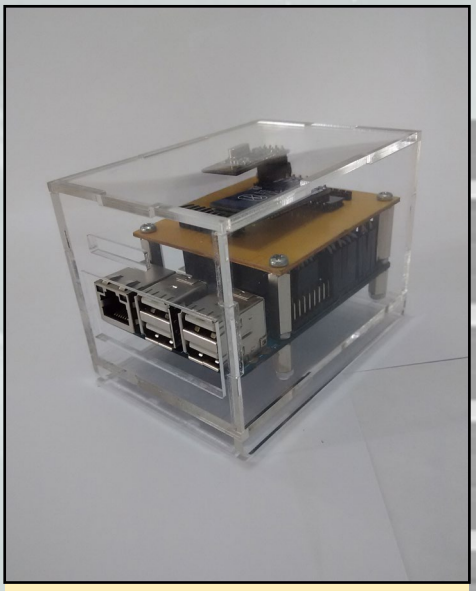


Figure 1 - ODROID-C1 in a clear case

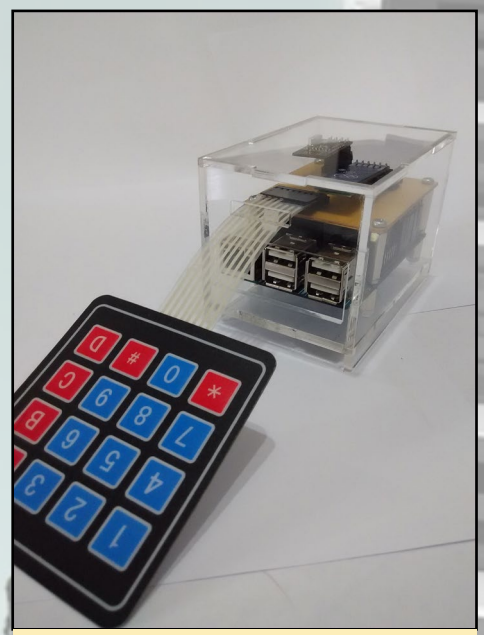


Figure 2 - ODROID-C1 with keypad

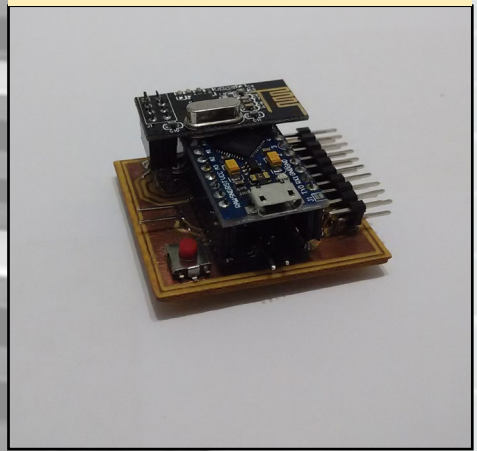
the position of the key that should be pressed. We used the smallest vibration motors that we could find in order to minimize any discomfort to the student while playing with this device.

The vibration motors are controlled by a microcontroller. We choose to use the Arduino Pro Micro because of its small size, and for the number of libraries available to simplify software development. Another transceiver module is used to connect the hand modules with the ODROID-C1 MPU wirelessly. All components of the Hand Module are powered by a single cell LiPo battery, boosted with a 5V step-up voltage regulator. We use a 180 mAh battery, which is small enough to fit in the small case, yet has enough capacity to supply the hand module for a reasonably usable time. All components except for the rings are packaged in a 3D printed case. A strap is attached to the case so that students can use the hand module on their wrist, just like a watch. Arjuna uses two hand modules, one for each hand.

How Arjuna Works

Arjuna communicates with keyboards by using the MIDI (Musical Instrument Digital Interface) protocol, a protocol which enables digital musical instruments, computers, and other supported devices to communicate with each other. With this protocol, the MPU can receive data, such as notes and velocity from the keyboard, while the student is playing.

Figure 3- ProMicro unit



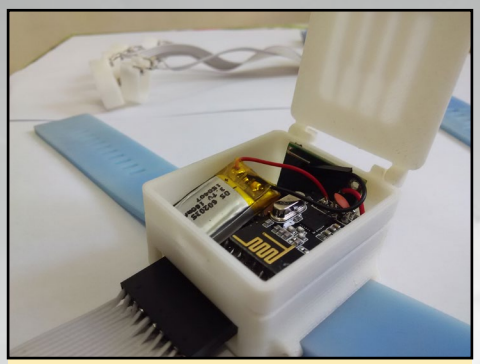


Figure 4 - Hands module top view

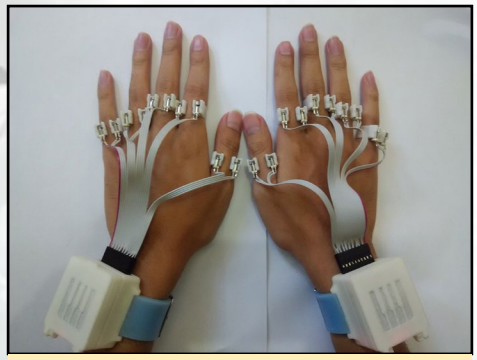


Figure 5 - Hands with modules attached

Figure 6 - Playing the keyboard

Arjuna has two modes: Listen Mode and Evaluation Mode. Both modes require two sets of data, one for the song, and one for the fingering data. Songs are stored in a MIDI file (.mid), which could be found at several Internet sites, or can even be your own song made with any MIDI song creator (and there a lot of them out there). The fingering data can be created from the MIDI song file with software we call MidiFGR. In this software, you can manually set the correct finger to play for each note, and it will then generate a new file. This file is needed for guiding the student in playing the correct key, with the correct finger.

In Listen Mode, the MPU will read the chosen song, and play the song thru the keyboard. At the same time, the MPU will send commands to the hand modules, which will vibrate the correct finger for every notes that should be played. In this way, the student can learn the proper fingering skill.

In Evaluation Mode, the student can play the keyboard, and receive guidance. While the student is playing, the MPU evaluates the data received from the keyboard by comparing it with the song file. And when the student presses the wrong piano key, the MPU will detect this and send a command to the hand module. This command will then vibrate the motor either on the left, or on the right side of the correct finger. The side will depend on which position the key that should be pressed is located, relative to the incorrectly pressed key. For example: If the correct key is located to the right of the wrongly pressed key, the left motor will vibrate. On the other hand, if the correct key is located to the left of the wrongly pressed key, the right motor will vibrate. Of course this behavior can be reversed in the MPU settings.

About The Arjuna Project

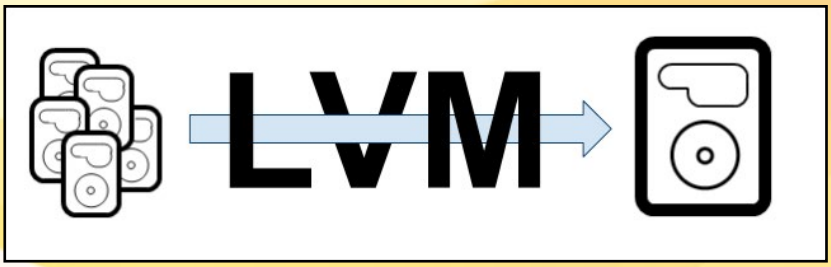
We are a three person team from Gadjah Mada University, Indonesia. Originally, this project was dedicated to helping the visually impaired in learning how to play the piano. Learning to play the piano is not easy for anyone, but for someone with limited vision capability, it is much more difficult. This device was created to introduce the visually impaired to some of the basic piano skills without the need for sight. With Arjuna, we hope to open a new window for the visually impaired, a window into the beautiful world of music.

Arjuna is open source. Currently we are rebuilding our public repository for easier installation, and it is available at http://github.com/ArjunaElins/Arjuna. Every source such as schematics, 3D files, and codes, are stored here.

LOGICAL VOLUME MANAGEMENT BASICS

WORK DIFFERENTLY WITH YOUR DISK SPACE FROM NOW ON

by David Gabriel



You can never have enough storage. We always dream of getting a bigger storage unit, such as a hard drive, or like that time when you were counting your pennies and decided to get a smaller eMMC for your first ODROID only to find out that what you want now is space: much more space. I'm talking to you, XBMC users!

At the most basic level, when you needed a bigger volume than your previous one, you could insert a new and larger disk on your machine, format it, copy your data to it and just be happy, which is simple as we have been taught from the beginning. It is what you do regularly, as we are nothing less than fond of having, in most cases, a 1TB USB drive plugged into our ODROIDs.

However, if you stop to think about it, you just committed an entire terabyte of data for a single purpose. Worry no more, I am here to teach you can do a lot more with your drive. From now on, you won't need to use all of your space on a single file system, or keep trying to guess the best partition scheme, or worse, reformat the disk, change the partition scheme and reinstall everything just because you bought a new disk.

First of all, backup your files. I know this may sound obvious, but most of us don't do a regular backup, right? We will be messing with space allocation, partitions, and file systems. So, even if you don't have bad fingers where everything you touch just blows up, it is always a good idea to just relax and know that if you arrive at the moment where you say "Whoops, where is all my data," you can say afterwards, "No problem, I have it all on my backup".

To proceed with these steps, you will need some free space. It can be any blank drive that you might have, or that old partition that you don't use anymore. All you need is a logical device on which to save your data.

Installation

That is the easy part. You just need to install lvm on your system:

```
$ sudo apt-get install lvm2
```

Configuration

Here is where the magic happens. If you have never heard of LVM before, you should know that the volumes are divided into physical volumes, volume groups and logical volumes. Physical volumes are your actual disk partitions. Volume groups are where all of your raw space is located, and logical volumes are equivalent to the normal partitions that you are used to. So, let's create our physical volume first. To do that, type the following into a Terminal window, where /dev/sda1 is the partition you want to "format" as LVM. This makes it available for use on our new LVM.

```
$ pvcreate /dev/sda1
```

Then, we create the volume group, where rootvg is just a label you give to the volume group, and /dev/sd1 is the physical volume we just created.

```
$ vgcreate rootvg /dev/sda1
```

By now, you can see the volume group created. Just type:

```
$ vgs(information)
```

or

```
$ vgdisplay(attributes)
```

Now we have to create logical volumes so that we can use the space:

```
$ lvcreate -L 10G -n homelv
```

This will create a new logical volume with 10 gigabytes named homelv. Note that this will create a new device under /dev label dm-0, which will internally point to the actual physical disk /dev/sda. It also creates a link under /dev/mapper/rootvg-homelv to the /dev/dm-0 so that you won't have to remember numbers in the future. To check the status of the logical partitions, type:

```
$ lvs(information)
```

or

```
$ lvdisplay(attributes)
```

From now on you can proceed just like any other method. Create a new filesystem and mount it in order to start saving data to it. Remember to use the device created previously:

```
$ mkfs -t ext4 /dev/mapper/root-
vg-homelv
$ mount /dev/mapper/rootvg-homelv
/home
```

Now you have your disk up and running with LVM. But you will only see the difference from LVM to normal partition when you need to change it.

After having been set up LVM for some time and having a lot of users logging and saving their files under /home directory, it will eventually get full, unless you increase it. How? First, unmount your file system:

```
$ umount /home
```

Increase the logical volume first:

```
$ lvextend -L +10G /dev/mapper/
rootvg-homelv
```

This will give you an extra 10 gigabytes on the logical volume. However, we still have to increase the file system to match with the logical volume. You can do that with:

```
$ e2fsck -f /dev/mapper/rootvg-
homelv
$ resize2fs /dev/mapper/rootvg-
homelv
```

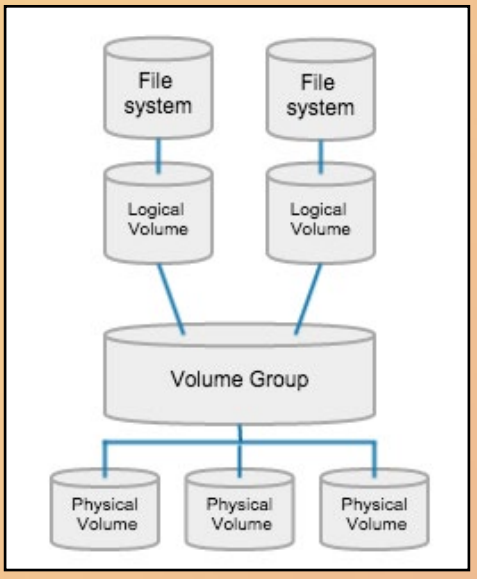
The first command will check for any inconsistencies on the file system, and the second will resize it with new logical volume size. After some time, you also need a new file system in order to install an application. However, you realize that you gave all the space that you had left on the volume group to the homelv. What now? Just add a new partition to it:

```
$ vgextend rootvg /dev/sda2
```

You can add as many partitions/disks as you want to the volume group in order to create a big pool of free space. After that, just repeat the steps to create/extent logical volumes and file systems as you wish.

Now you know how to make the experience of creating partitioning schemes a lot easier. One thing to note is that you can do this with all of your file systems. Just create a temporary mount point, move all of your data there, and then adjust /etc/fstab to the right devices. It is important to note that you can't tamper with /media/boot, since that is where the bootloader looks for the first files in order to send them to RAM and actually boot the system. All of this happens long before the lvm services are initiated, so it will make your system unbootable if you do so, so be careful.

These are the basics to set up LVM on your system. There is a lot more that you can do with it, but this should be enough to start experimenting with it and discovering all of its possibilities.



QT5 DEVELOPMENT

BUILDING A SINGLE APPLICATION USER INTERFACE

by Christopher Dean

This article describes how to use the Qt5 programming language and the Qt5 Developer's Dream image in order to create a Single Application User Interface. This is helpful for individuals building kiosks, media players, medical equipment, and other embedded Linux systems. We will quickly jump into creating a simple Qt5 application, and then learn how to configure the session to run the application on startup.

Getting started

The first thing is to download and flash the latest Qt5 Developer's Dream image to your SD card or eMMC module. Download links for the image may be downloaded from http://bit.ly/1PMDwKP.

Figure I - Qt5 Developer's Dream desktop

Sample application

After booting into the image, we can now create our first Qt5 application! Open Qt Creator from the application drawer at the top left of the Lubuntu desktop environment by selecting Menu -> Programming -> QtCreator. After QtCreator launches, choose File -> New File or Project from the action

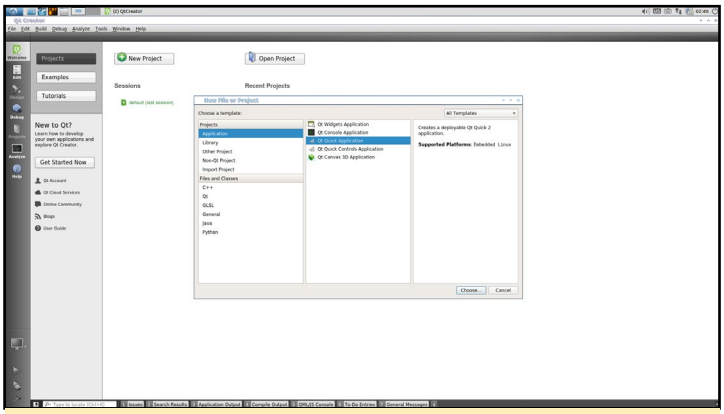


Figure 2 - Creating a new project

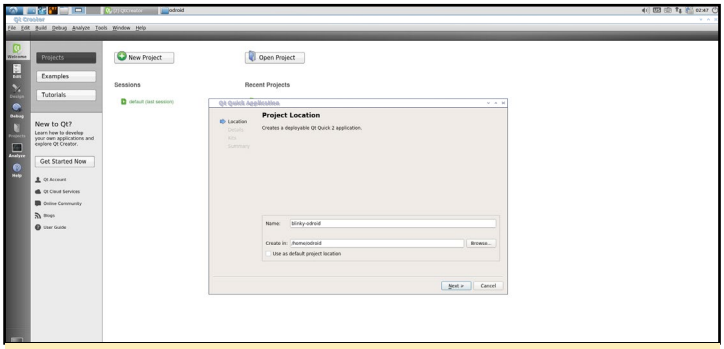


Figure 3 - Naming the project

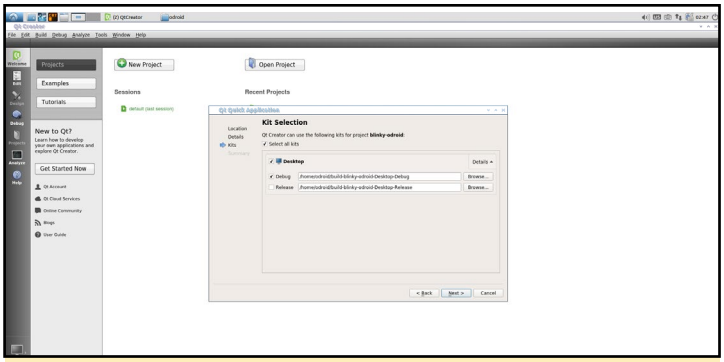


Figure 4 - Choosing the Desktop kit

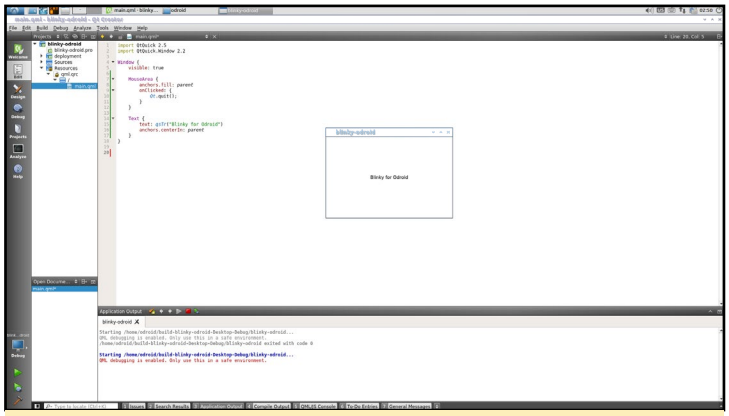


Figure 5 - Changing the text element to "Blinky for ODROID"

bar, then choose Qt Quick Application. Next, give the project a name and choose the directory, and configure the Qt5 project using the "Desktop" kit.

Continue through the dialogs until the project is created. Then, under the projects dock on the left, navigate and open Resources -> qml.qrc-> / -> main.qml. After opening the file, change the text inside the Text element to from "Hello World" to "Blinky For ODROID". Then, build and run the project by pressing Ctrl + R. After clicking the application to close it, open a terminal and navigate to the projects directory. At the top of the project directory, run the following command in order to clone the QGPIO class:

```
$ git clone http://github.com/Tpimp/qgpio.git
```

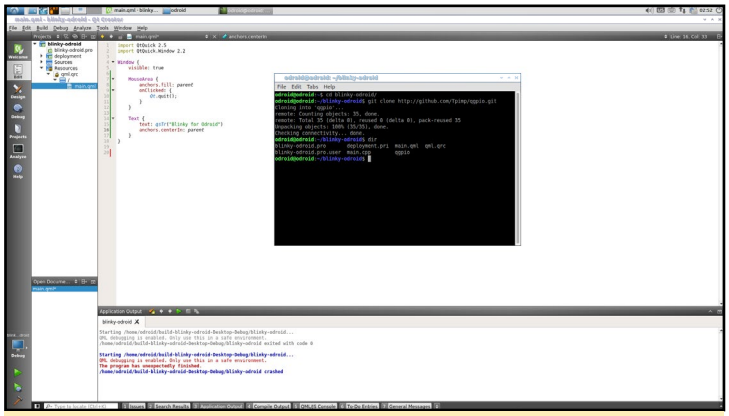


Figure 6 - Cloning the QGPIO project

Go back to the QtCreator application and right-click "blinky-ODROID" in the project dock, select "Add Existing Directory…" from the context menu, then choose the directories as shown in Figure 7. After adding the directory from the projects dock, navigate and open Sources -> main.cpp. Alter it to look like Figure 8. Note that this step is related to using QGPIO, not writing a general Qt5 application.

Next, we can make the "Blinky" part of the application. Re-open the main.qml file for editing.

After importing the com.embedded.io module defined in

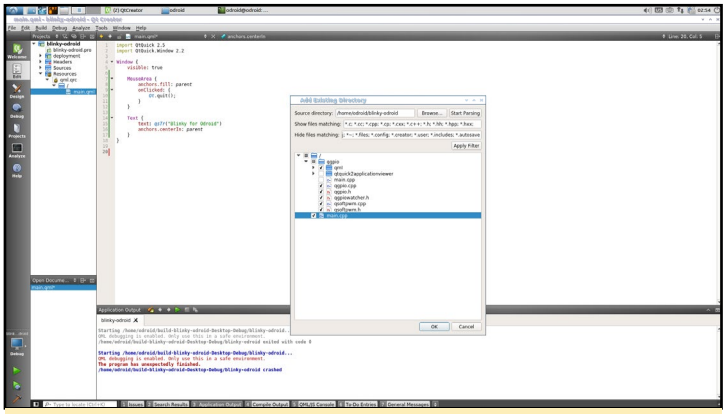


Figure 7 - Adding the QGPIO class to the project

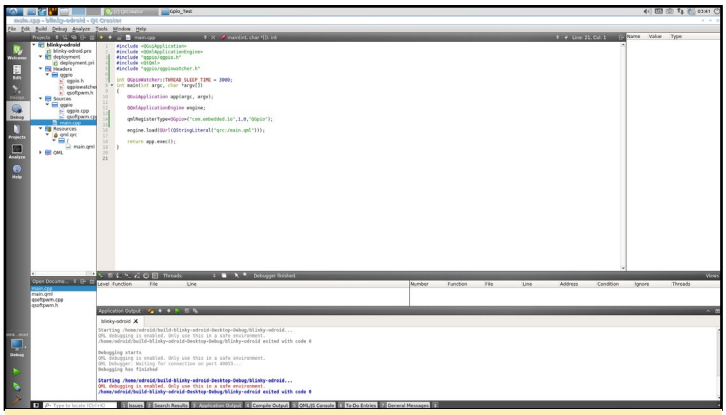


Figure 8 - Modifying the QGPIO class

the C++, create a QGpio element and set the ID to gpio200 and the number to "200". Next, create the function callback that will be invoked when the component is complete. In the function, set the GPIO direction to Out, set the edge to Falling, set active_low to false, and the value to false. Then, in the MouseArea object, replace the onClicked function statements to simple toggle the GPIO value:

```
gpio200.value = !gpio200.value;
```

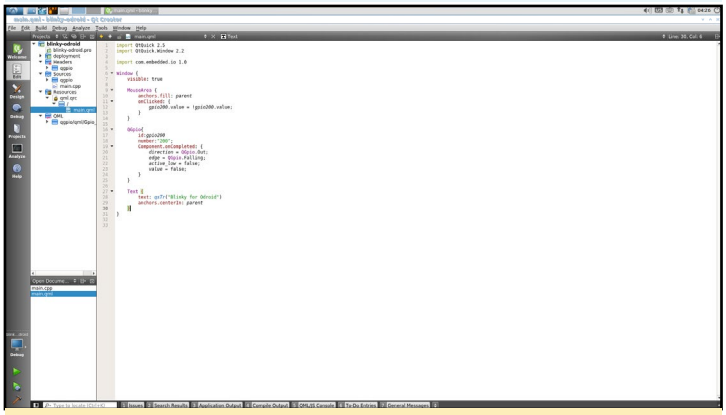


Figure 9 - Modifying the main.qml file

Run the program and click on the center, then observe the value of GPIO 200 changing by typing the following command into a Terminal window:

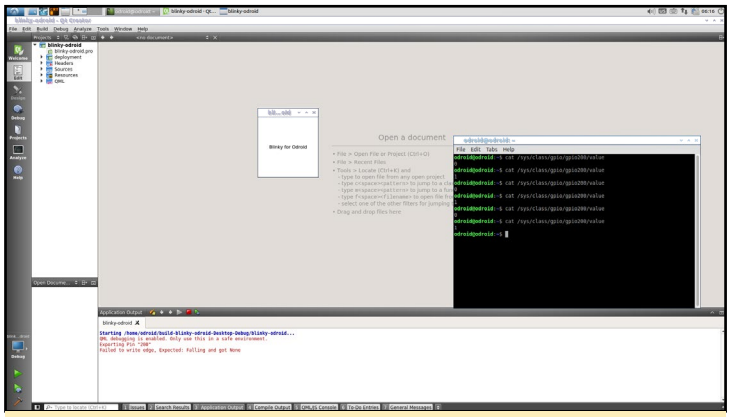


Figure 10 - Checking the value of GPIO 200

```
$ cat /sys/class/gpio/gpio200/value
```

Now you are ready to blink! Connect a low amperage LED to pin 200 and ground, as shown in Figure 3. If you do not have a low amperage LED, use a resistor or a transistor with a separate power source. The scope of LEDs and GPIO voltage are too large for this article, but make sure not to overdraw your GPIO port with a big LED! Wire your LED, then run the application. The program should then flash the LED when clicking in the center. The next step is making the Blinky application our startup X11 application, which definitely is an overly simple task for an ODROID, but is fun for demonstration purposes.

Figure 11 - Blinky-ODROID wiring diagram

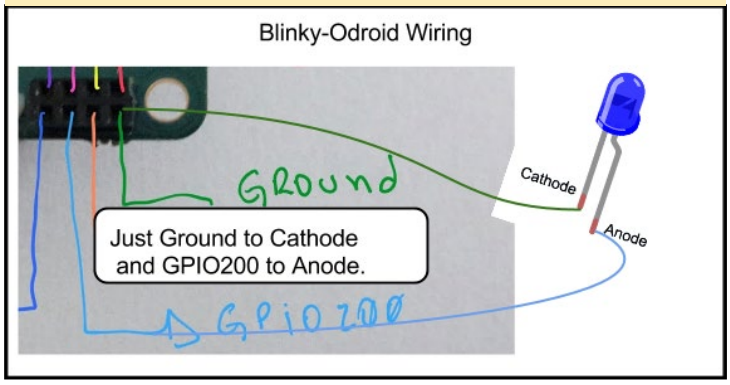
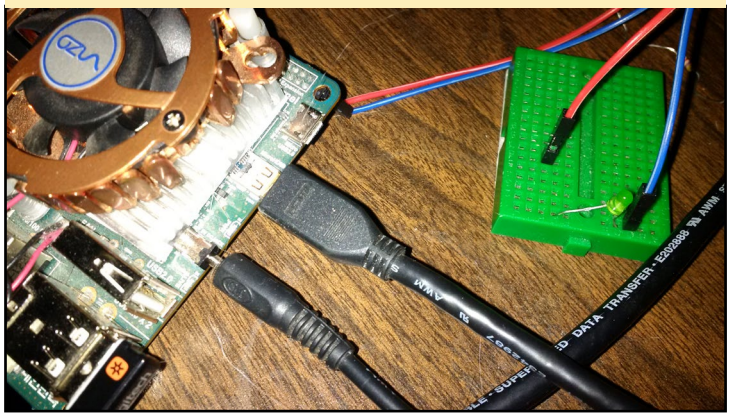


Figure 12 - Blinky-ODROID wiring example



Create a desktop configuration

There are many ways to configure X11 to startup with just a single application. Technically, we should set the default Desktop Environment to be "Blinky" instead of Lubuntu and remove the other option to ensure the user boots only into "Blinky". However, creating the desktop configuration file allows us to switch back and forth between testing our application and the development environment. Create a blinky.desktop Xsession configuration file in the /usr/share/xsessions directory as shown in Figure 13. The next step is to compile the blinky-ODROID application and place it into /usr/bin/. However, if we run blinky-ODROID in its current form, we cannot logout, which happens on the program exit. Before testing the Blinky desktop environment, we should make some changes.

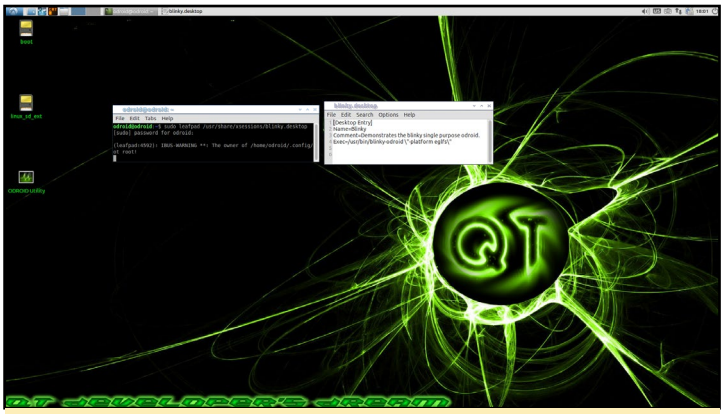
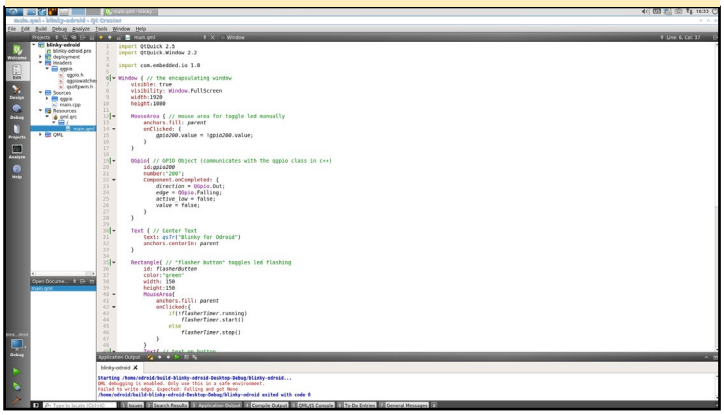


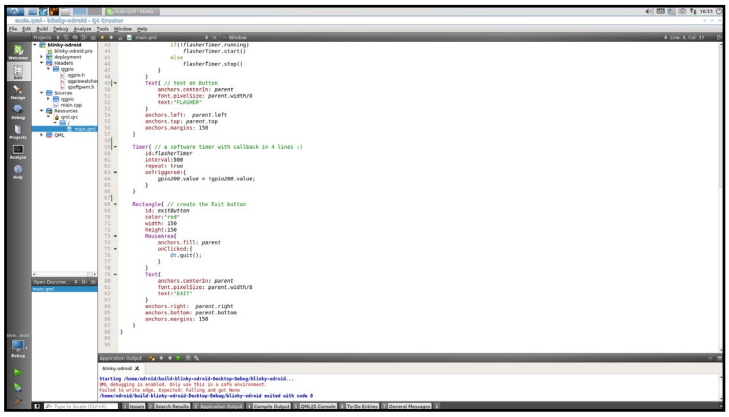
Figure 13 - Example blink.desktop Xsession configuration file

Adding features

First, we need to add some code to make our blinky demonstration work automatically, as well as manually. First, add the width and height to match your current resolution, then add the flasher button, which is simply a rectangle with another MouseArea in it. The mouse click will then start and stop a timer, which will be defined in main.qml, as shown in Figure 14.

Figures 14 and 15 - Blinky-ODROID interface code





The logic is that if the timer is not running, then we start it, else we stop it. First, give the rectangle a height, a width, and a color. Add text to the button and anchor it to a location on the screen. Below that, create the "flasher timer" and give it an interval of 500ms, which is equivalent to half of a second. The timer is then set to repeat, and an "onTriggered" callback is created, which states "while running, toggle GPIO 200 every half second."

Finally, add an exit button, which is very similar to the flasher button, but colored red and located in the bottom right. Also, instead of toggling the timer, the exit button will close the application. If this was a real replacement for the desktop environment, we would shut down instead of quit. Quitting brings you back to the LightDM screen in order to choose where to go next.

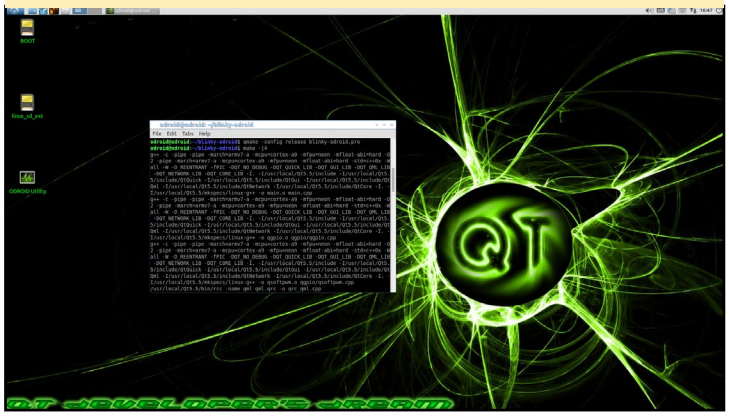
Creating the binary

Until now, QtCreator has likely been performing a shadow build and operating in "debug" mode. When deploying the application, we want to build for "release". Open a Terminal window and navigate to the project directory. Once in the directory, run the following commands:

```
$ qmake -config release blinky-ODROID.pro
$ make -j4
```

It should build without any issues. If you encounter errors, download the source from http://bit.ly/1KsPX0z. After you build it, install it by copying the executable to the /usr/bin directory:

Figure 16 - Building the Blinky-ODROID binary



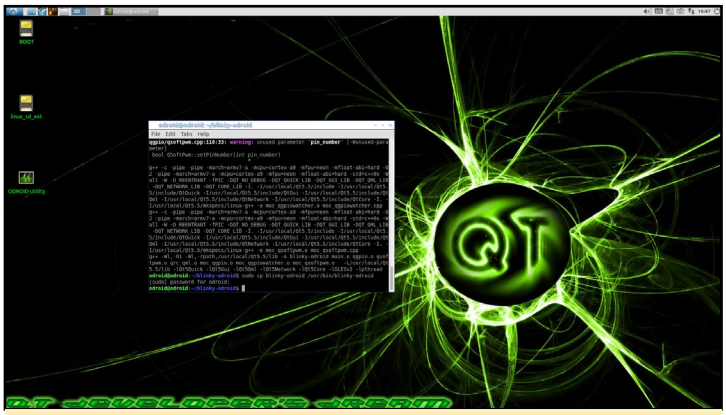


Figure 17 - Copying the Blink-ODROID binary to /usr/bin

```
$ sudo cp blinky-ODROID /usr/bin/blinky-ODROID
```

Testing Blinky

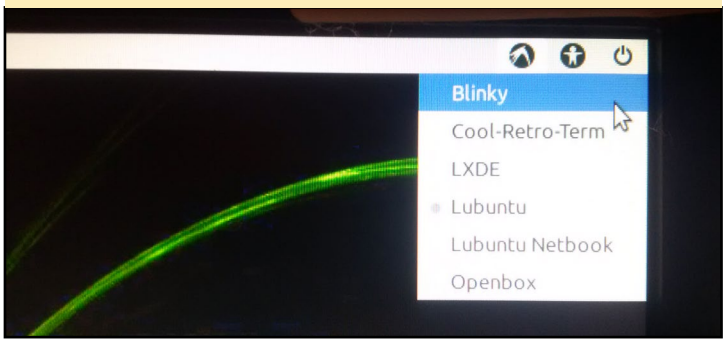
Testing is very easy after creating the xsession configuration file. First, log out of the Lubuntu session with the normal log-out button. You should be back at the LightDM screen waiting to login. At the top right, click the icon in order to open the Desktop Environment list, and choose "Blinky". Then, log in with the normal ODROID/ODROID credentials. If all went well, you should see the blinky-ODROID example appear on the screen. This was a very simple project, but you can use it to build very complex desktop environments. The largest drawback is the lack of a WiFi manager and other convenient settings dialogs.

Further reading

If you want to learn more about Qt5 or building desktop environments, here are some links:

Desktop Environments: http://bit.ly/1MyywH5
Embedded Linux: http://bit.ly/1LMrksN
Qt5 Desktop Services: http://bit.ly/1fWjtxk
Plasma 5 Desktop Environment: http://bit.ly/1EAUtax

Figure 18 - Selecting the Blinky desktop environment using LightDM



SAMIIO

EASILY BUILD POWERFUL I/O APPLICATIONS

by Venkat Bommakanti

About a year ago, Samsung introduced a secure open-cloud based data exchange platform, called Samsung Architecture Multimodal Interactions IO (SAMIIO). Using this platform, one can develop applications and services utilizing simple open APIs and SDKs to send, receive and visually examine data of diverse types. These applications can written using languages such as Python, Ruby, Javascript, and PHP.

Through a real-world example, this article illustrates the simplicity of the SAMIIO platform, by using an ODROID-C1+ and a Python application in order to publish weather data collected through an ODROID-SHOW and an ODROID Weather Board.

Requirements

- An ODROID-C1+. Although this article utilizes a C1+, the technique can apply to any ODROID device that is compatible with the ODROID-SHOW and Weather Board.
- ODROID-SHOW & Weather Board
- C1+ accessories such as an HDMI cable, CAT 5E+ ethernet cable or WIFI 3 adapter, Bluetooth adapter module 2 (BT 4.0+), recommended PSU, RTC battery, and ODROID-VU or 3.2" Touchscreen
- A 16GB+ eMMC 5.0 card with latest C1+ specific Lubuntu desktop image and/or an 16GB+ Class 10 MicroSD with an SDCard reader/writer
- A network where the device has access to the internet and the ODROID forums
- Networked access to the C1+ via utilities like PuTTY, FileZilla, TightVNC Viewer (MS Windows 7+), terminal (Mac, linux), etc., from a testing desktop

System setup

The setup is shown in Figure 1.

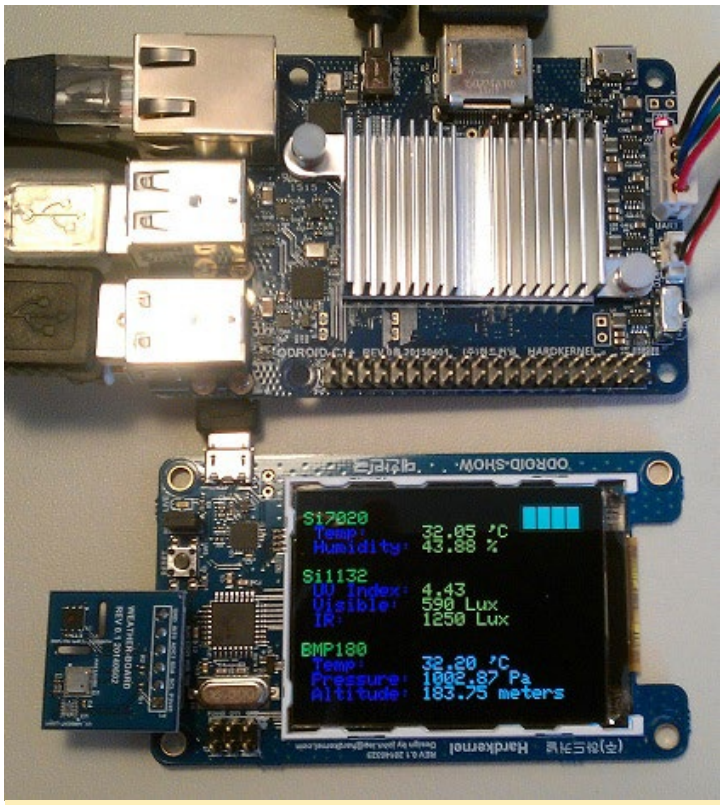


Figure I: C1+ setup for SAMIIO

Install Lubuntu

Install latest C1+ image on to the eMMC card. Attach the eMMC card to the C1+. With the VU display attached, boot up the system. Run the ODROID Utility program and set the display resolution to say 800p. Reboot, then expand the installation partition to use the entire eMMC by selecting the "Resize your root partition" option. Reboot and re-run the ODROID Utility again, configuring and updating all remaining relevant aspects of the system, rebooting afterward. Ensure you are always logged in as the default "odroid" user, unless otherwise specified. For the most recent images, you would have to run the following commands in order, to update the

system:

```
$ sudo apt-get autoremove && sudo
apt-get update
$ sudo apt-get dist-upgrade &&
sudo apt-get upgrade
$ sudo apt-get install linux-
image-c1
```

The last command is to fetch the latest C1+ kernel, just in case it was held back. Shutdown the device, then attach all the accessories and cables to the C1+, including the SHOW and Weather Board accessories. Reboot. Check the USB port associated with the ODROID-Show with the following command:

```
$ ls -lsa /dev/ttyUSB*
0 crw-rw---- 1 root dialout 188,
0 Aug  12 10:25 /dev/ttyUSB0
```

Check the system version from a terminal to ensure you have the latest (as of this writing):

```
$ uname -a
Linux c1-1 3.10.80-122 #1 SMP
PREEMPT Mon Aug 10 20:27:04 BRT
2015 armv7l armv7l armv7l GNU/
Linux
```

Update Weather Board sketch

While the weather board sketch is not directly needed to update the SAMIIO website with the weather data, it is useful to visually validate the accuracy of the data being submitted. Shown below are the steps to upload a modified sketch, that make the presented data well organized, so one can quickly find the data being submitted. Install the Arduino IDE for C1+, if not already present.

```
$ sudo apt-get install arduino
$ cd ~ && mkdir zBU && cd zBU &&
mkdir sami && cd sami
$ git clone https://github.com/
hardkernel/ODROID-SHOW.git
$ cd ODROID-SHOW/weather_board
```

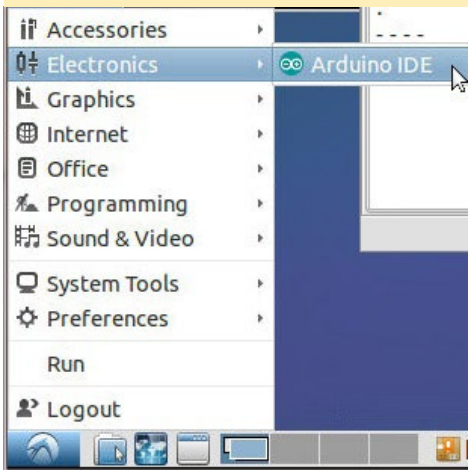
Ensure that the weather_board.ino sketch file is present at ~/zBU/sami/ODROID-SHOW/firmware/weather_board/weather_board.ino

Create an empty patch file called weather_board-ino-changes, in the current directory, copy the following contents and save it:

```
20c20
< const char version[] = "v1.3";
---
> const char version[] = "v1.4-
vb";
74c74
<             tft.setCursor(250,
200);
---
>             tft.setCursor(200,
200);
88,89c88,89
<             tft.println("Temp :
");
<             tft.
println("Humidity :");
---
>             tft.println(" Temp:
");
>             tft.println(" Hu-
midity:");
94,96c94,96
<             tft.println("UV In-
dex : ");
<             tft.
println("Visible :");
<             tft.println("IR
:");
---
>             tft.println(" UV
Index: ");
>             tft.println(" Vis-
ible:");
>             tft.println("
IR:");
101,103c101,103
<             tft.println("Temp :
");
<             tft.
println("Pressure :");
<             tft.
println("Altitude :");
---
>             tft.println(" Temp:
");
>             tft.println(" Pres-
sure:");
>             tft.println(" Alti-
tude:");
246c246
<     tft.setCharCursor(7, 11);
---
>     tft.setCharCursor(11, 11);
248c248
<             tft.println(" *C
");
---
>             tft.println(" 'C
");
264c264
<     tft.setCharCursor(7, 2);
---
>     tft.setCharCursor(11, 2);
266c266
<             tft.println(" *C
");
---
>             tft.println(" 'C");
282c282
<     tft.setCharCursor(10, 7);
---
>     tft.setCharCursor(11, 7);
287c287
<     tft.setCharCursor(5, 8);
---
>     tft.setCharCursor(11, 8);
```

Update the sketch with this patch file

Figure 2: Launch Arduino IDE



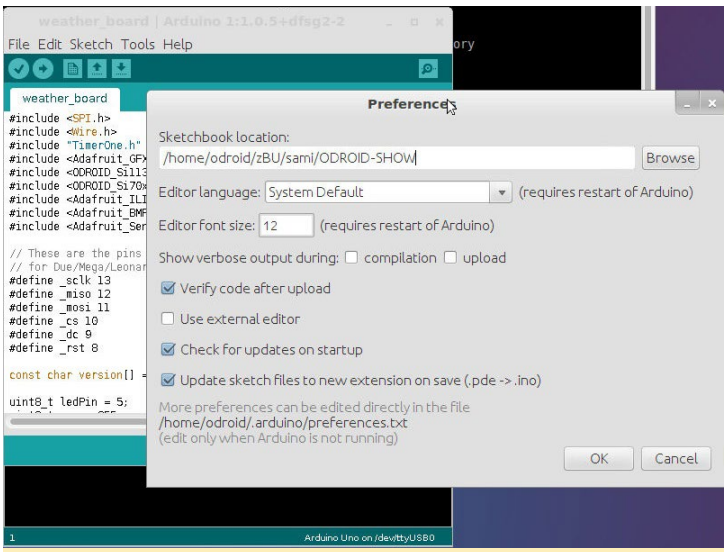


Figure 3: Arduino preferences

using the command, then launch the Arduino IDE as shown in Figure 2. Select the Preferences menu item under the File menu, and select the ODROID-SHOW folder as the sketch folder, as shown in Figure 3.

Figure 4: Arduino libraries location

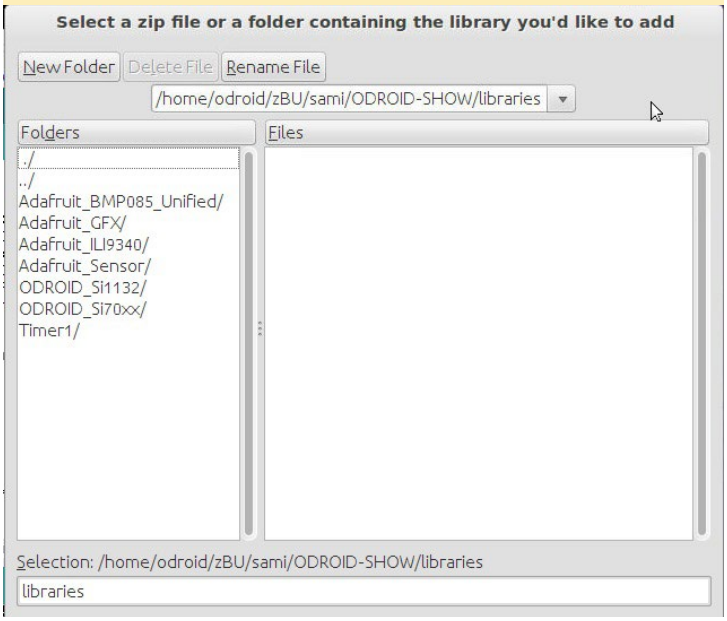
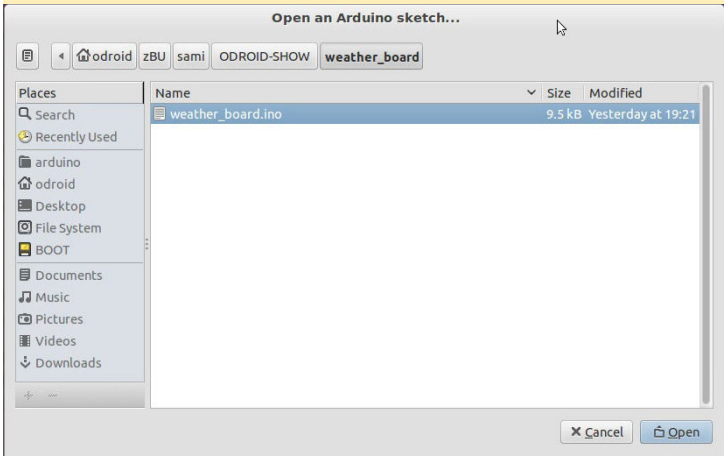


Figure 5: Open sketch




```
$ patch -input weather_board.ino patch.txt
```

To add the proper libraries, Select the Sketch → Import Library → Add Library menu options and point it to the ~/zBU/sami/ODROID-SHOW/libraries folder as shown in Figure 4. Open the sketch using the File → Open menu items as shown in Figure 5.

The sketch (code) should appear in the window. The jumper on the ODROID-SHOW (near the Reset button) must be installed. Also, select the serial port using the Tools → Serial Port → /dev/ttyUSB0 menu items. Upload the sketch by clicking the Right Arrow circular icon. The upload progress will be visible as shown in Figure 6.

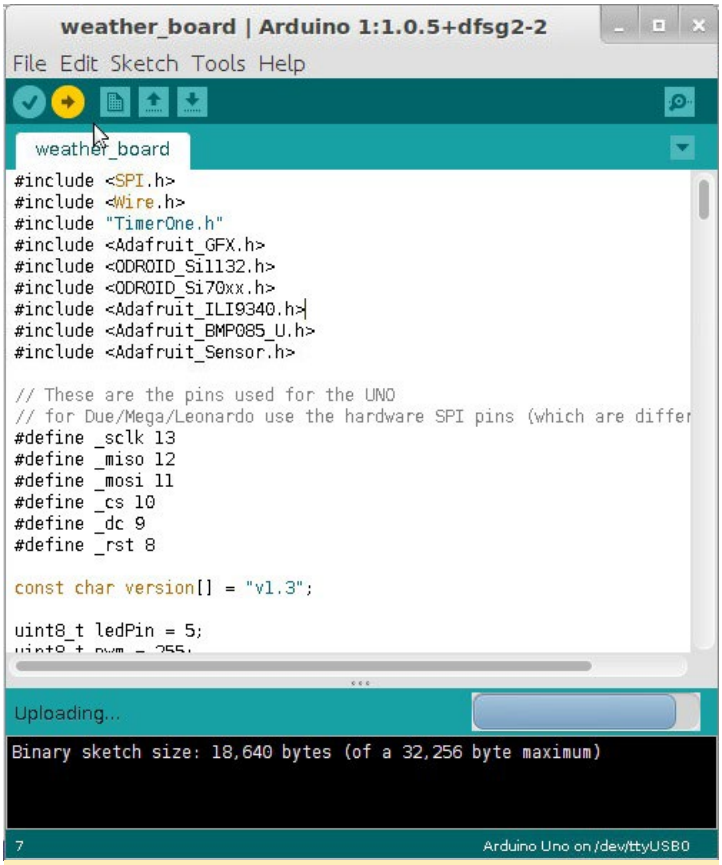
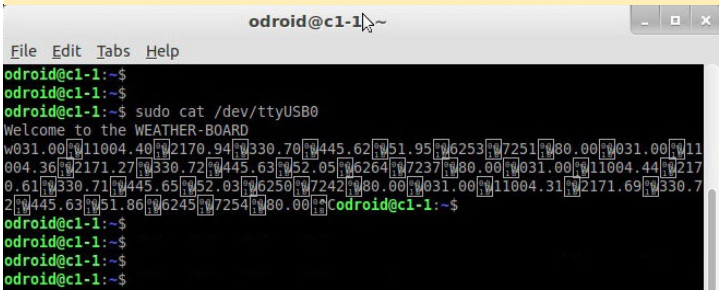


Figure 6: Sketch upload progress

You should start observing the ODROID-SHOW displaying the weather data. You can also check to see if you can access the output from the command line, as shown in Figure 7.

Figure 7: Raw output on /dev/ttyUSB0



The output format is useful to write the Python application to submit the weather data to the SAMIIO website.

SAMIIO account and device setup

Details of the SAMIIO platform can be found at the main SAMIIO website. Access the SAMIIO account setup website at http://bit.ly/1ds2ONj. You should see a page as shown in Figure 8. Since a user account has not been setup yet, we cannot login yet. Click on the Sign up here link and you should see the page as shown in Figure 9.

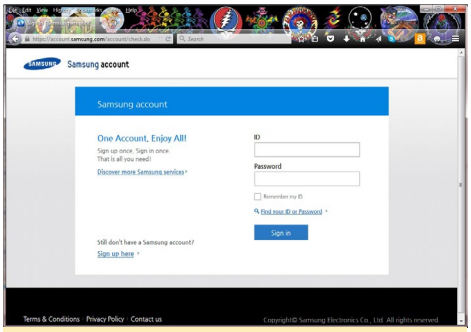


Figure 8: SAMIIO account access

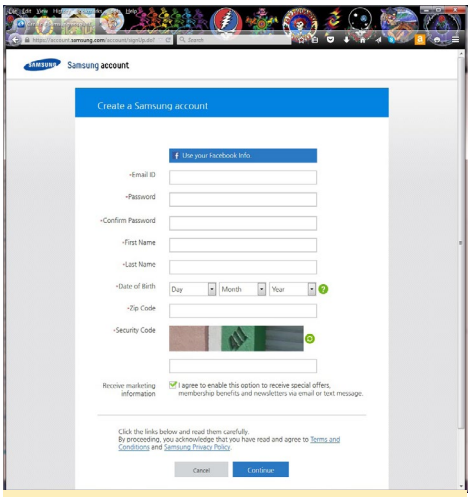


Figure 9: SAMIIO account setup

Enter information relevant to your account and click the Continue button. Get back to the login screen as shown in Figure 8, enter the login info click the Sign In button. You will be presented a page as shown in Figure 10:

Enter the phrase Temp Sensor in the field that has a grayed-our prompt: Start typing the name of your sensor. A new

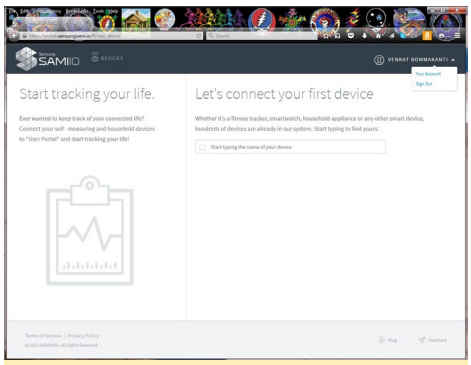


Figure 10: Post-login screen

It will take you to a new page as shown in Figure 12.

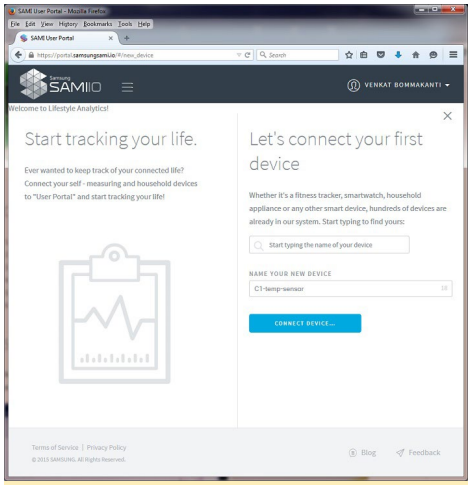


Figure 11: Device creation

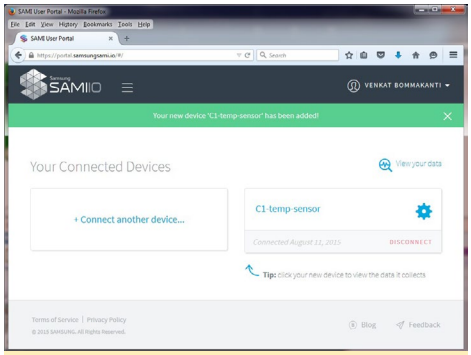


Figure 12: Device main screen

Click on the blue gear icon to create some IDs useful in the communication process. You will be presented with a page that looks like Figure 13. Note the Device ID that is generated by the system. Click on the link that has the caption: GENERATE DEVICE TOKEN. You will be presented with the screen as shown in Figure 14. Note the Device Token that is generated by the system. The device ID is especially used in the

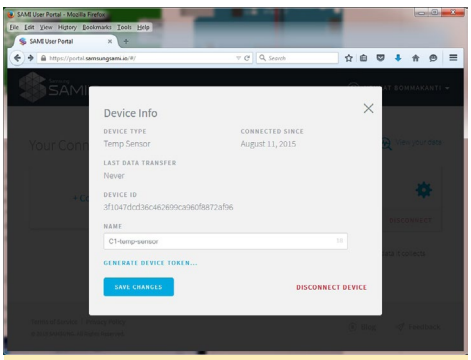


Figure 13: Device ID

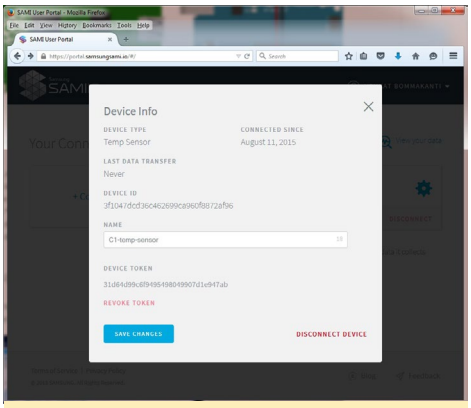


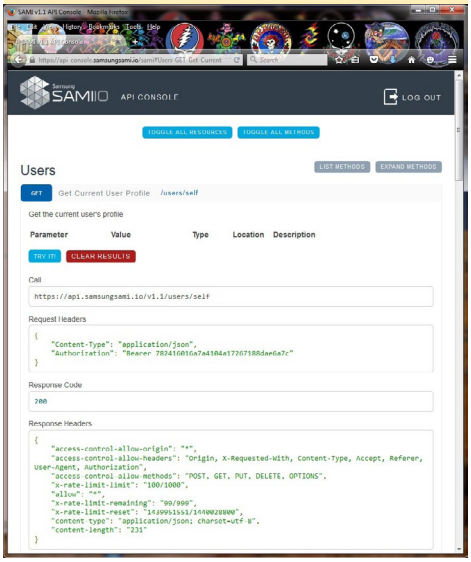
Figure 14: Device Token

communication process.

The SAMIIO website provides a mechanism to obtain additional device data (bearer-id for authentication) and test the SAMIIO API to submit data. Before writing an application to submit the data, the API can be tested via this test website.

Access the API test site at http://bit.ly/1NeVs1I. Login to the site using your credentials that you setup earlier. Under

Figure 15: User information



the Users set of API, click the Get Current User Profile link. You will be presented with a temporary page with the TRY IT blue button. Click it and it will produce output as shown in Figure 15.

The bearer-id (for authentication purposes) is useful in submitting the weather data through an application. Next, traverse the page to the Messages section. Enter some sample temperature data in json format, as shown in Figure 16. Use the Device ID obtained earlier and enter the integer value (# of milliseconds since Jan 1 1970) of current time for the ts field. Click the TRY IT blue button. You will be presented with the screen as shown in Figure 17. Note the response body reflecting the successful submission of temperature data, with a message response id.

Figure 16: Messages API input

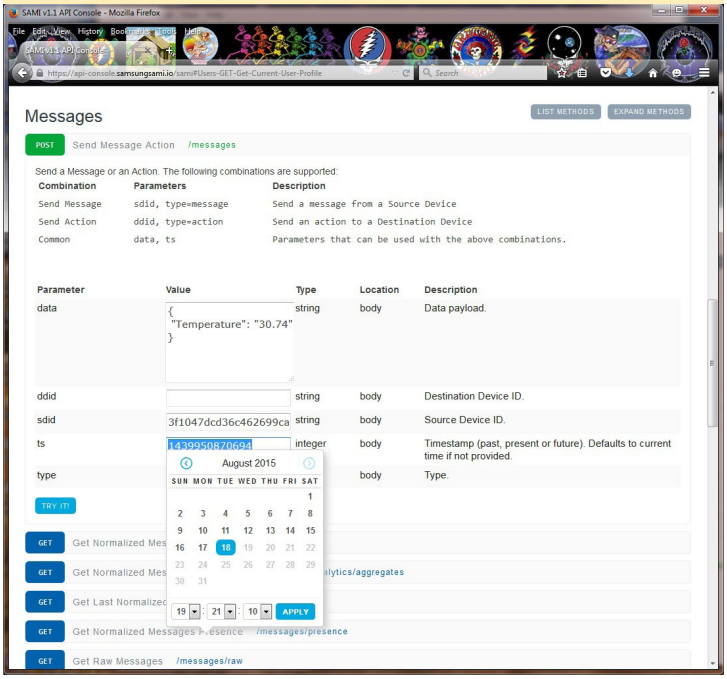
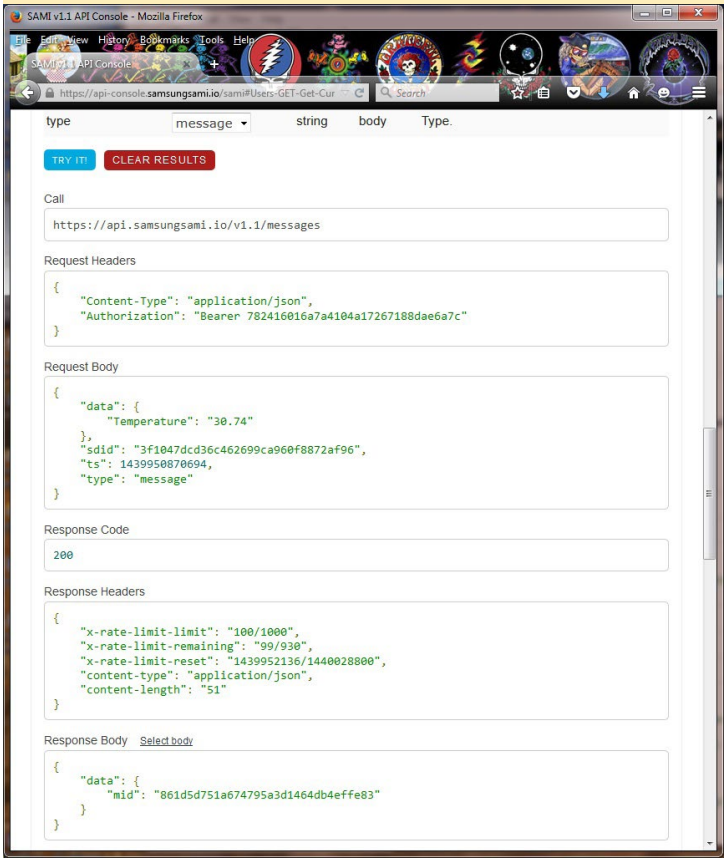


Figure 17: Messages API output



SAMIIO Python prerequisites

At present, we have Python 2.7.6 installed on the standard C1 images. Versions of Python earlier than 2.7.9 have restrictions in their ssl module that limit the configuration that urllib3 can apply. In order to address this issue, install the following components:

```
$ sudo apt-get install libffi-dev libssl-dev
$ sudo apt-get install python-pip python-dev build-essential
$ sudo pip install pyopenssl ndg-httpsclient pyasn1
```

The use of the urllib python package results in verbose code. Install the requests package to simplify the Python code using the commands:

```
$ cd `/zBU
$ git clone git://github.com/kennethreitz/requests.git
$ cd requests
$ sudo python setup.py install
```

Sample SAMIIO aware application

Given the information obtained so far, we can create a python script called sami-req-client.py in the folder: ~/zBU/sami/. Ensure that it has the permit execution attribute (chmod 755) set. Following is the content of that script. Take a few minutes to see how information gathered so far, has been used in the script:

```
#!/usr/bin/python

#
# sami-req-client: Util to post weatherboard info
# to Samsung's SAMI service.  Presumes
# weatherboard works off ttyUSB0.
#
# (c) Venkat Bommakanti
# 08/05/15, CA, USA
#
# Free to use at your own risk.  No warranties implied.
# import sys, time
```

```
import serial, json, requests

# SAMI service & device info
sami_url = "https://api.sam-
sungsami.io/v1.1/messages";

# bearer: device-token: use your
bearer token here
bearer = "Bear-
er 75bced8e1e456095673d-
a70f375e187a1";

# sdid: device-id (source-id); :
use your device token here
sdid = "3f1047dcd36c-
345678760f8872af96";

# weather board ttyport: check on
your setup
weather_board_ttyport = "/dev/
ttyUSB0";
ttyport_speed = 500000;

# headers
headers = {
      "Content-type": "applica-
tion/json",
      "Authorization": bearer
}

# initialize
close_ok = True

#
# We'll be reading only 8 chars
off tty port.
# The data (char) pattern will be
something like:
#   wnabcdefg
# where 'wn' could range from
'w0' to 'w8', and
#   'abcdefg'
# could be a string equivalent of
any number between
#   '00000.00' and '99999.99'
# and any integer in between.
#

# 1 char length
A_CHAR = 1

# to clear a 10-char buffer
cl_phrase = ['\0', '\0', '\0',
'\0', '\0', '\0', '\0', '\0',
'\0', '\0' ]

# place holders
EMPTY_STR = ""
temp_val = EMPTY_STR
pres_str = EMPTY_STR
alti_str = EMPTY_STR
time_val = 0

try:
    iter = 0

    # setup serial port
    ser = serial.Serial(weather_
board_ttyport, ttyport_speed,
timeout=1)
    print "\n0: TTY-port: " +
ser.name
    ser.close()

    while True:

       if (close_ok):
              # open device to
read weather data
              sys.stdin =
open(weather_board_ttyport, 'r')
              close_ok = False
              print "\n1: opened
tty"
              # warning: no
timeout, could hang.
              # wait 10 secs for
init
              time.sleep(5)

              # first get welcome
lines
              wl_line = sys.
stdin.readline()
              print "\n2: " +
wl_line + "\n"
              # now start read-
ing data
              data_coll = 0
              iter = iter + 1
              wb_char = sys.
stdin.read(A_CHAR)
              print "\n3: " +
wb_char + "\n"
              while ((wb_char ==
'w') and (data_coll < 3)):

                     # clear
buffer that will hold weather
board data
                     wb_phrase =
cl_phrase

                     # read all
chars for a given piece of data.
                     # '\''
(decimal 27) is delimeter
                     idx = 0
                     while
(ord(wb_char) != 27):
                            wb_
char = sys.stdin.read(A_CHAR)

print "\n4: " + wb_char + ", ord:
" + \
      str(ord(wb_char)) + "\n"
                            wb_
phrase[idx] = wb_char
                            idx
= idx + 1
                            if
(idx > 8):

 break
                     # null ter-
minate
                     if (idx >
0):
                            wb_
phrase[idx-1] = '\0'
                     print "\n5:
" + "".join(wb_phrase) + "\n"

                     # typically
data arrives in order:
                     #   w0, w1
and then w2
                     print "\n6:
" + str(wb_phrase[0]) + "\n"

                     # skip the
1st char in buffer as it is
                     # just a
```

```
code for data type (temp, press, alti).
                        if (wb_
phrase[0] == '0'):
                                #
found start of weatherboard data - w0 -
 # for temperature

temp_str = "".join(wb_phrase[1:])

print "\n7: " + temp_str + "\n"

data_coll = data_coll + 1
                        elif (wb_
phrase[0] == '1'):
      # found pressure data

 press_str = "".join(wb_
phrase[1:])

print "\n8: " + press_str + "\n"

data_coll = data_coll + 1
                        elif (wb_
phrase[0] == '2'):
      # found altitude data

alti_str = "".join(wb_phrase[1:])

print "\n9: " + alti_str + "\n"

data_coll = data_coll + 1
                        else:
                                #
ignore other pieces of data

 print "\n10: ignoring next piece
of data\n"

                        if (data_
coll < 3):
                                #
read next piece of data
                                wb_
char = sys.stdin.read(A_CHAR)

 print "\n11: getting next piece
of data\n"
                        else:
 # found all 3 pieces of desired
data.  ignore others.
                                #
time to quit this iteration

print "\n12: all 3 pieces of data
collected\n"

                # close tty read
                close_ok = True
                sys.stdin.close()
                print "\n13:
closed tty"

                # timestamp (int)
to be sent to SAMI service
                time_val =
int(time.time()*1000)
                print '\n14: iter:
' + str(iter) + ", " + str(time_
val) + ', \
Temp: '  + temp_str + ', Press: '
+ press_str + ', \
Alti: ' + alti_str + '\n'

                #temp_str[5] is
bogus non-printable char
                [c + '\0' for c in
temp_str]
                temp_val =
float(temp_str[0:4])
                print '\n15: tem-
perature val: ', temp_val

                # prepare rest api
data params per SAMI service API
requirements
                sami_params = {

"sdid": sdid,

"ts": time_val,

"type": "message",

"data": {

 "temperature": temp_val

},
                                }
                # cleanup data
(long time val) to create good
json data
                jsami_params =
json.dumps(sami_params)
                print '\n16: jsa-
mi: ' + str(jsami_params)

                try:
                        print '\
n17: preparing to send data to
SAMI service'
                        req =
requests.post(url=sami_url,
data=jsami_params, \
headers=headers)
                        print '\
n18: sent data successfully !!!'
                except:
   # catch any exception
                        e = sys.
exc_info()[0]
                        print "\
n18: failed to send.  error: " +
str(e)

   # repeat search every 2 mins
                print "\n19:
waiting 2 mins to send next
data...\n"
                time.sleep(120)

       # repeat whole process
       print "\n20: next
teration......\n"

    # while block ends
except:
    # some error
    e = sys.exc_info()[0]
    print "\nErr: " + str(e) +
"\n"
    print "\nExiting...\n"

print "Done!"
```

Type the following to run the script:

```
$ cd ~/zBU/sami/
$ python ./sami-req-client.py
```

It should start running and gather the weather data from the ttyUSB0 port and submit it to the SAMI website, once every iteration, and submits temperature values once every 2 minutes. You can vary the temperature by moving the sensor closer to or away from a hot light source, such as a desk lamp. Figure 18 illustrates the temperature chart as seen from the SAMIIO website at http://bit.ly/1IVhfoZ. The website logs of the submitted data are shown in Figure 19.

The above script can be modified to show additional data such as the atmospheric pressure and altitude obtained from the weather board. The SAMIIO platform also provides SDKs for a variety of programming languages, which can be used to develop applications.

Additional resources

http://bit.ly/1O9FKFZ
http://bit.ly/1ds2ONj
http://bit.ly/1LUqbPP
http://bit.ly/1KBbpjQ
http://bit.ly/1NeVs1I
http://bit.ly/1IVhfoZ
http://bit.ly/1g6FbPj
http://bit.ly/1IVhcJI

Figure 18: Temperature chart on SAMIIO website

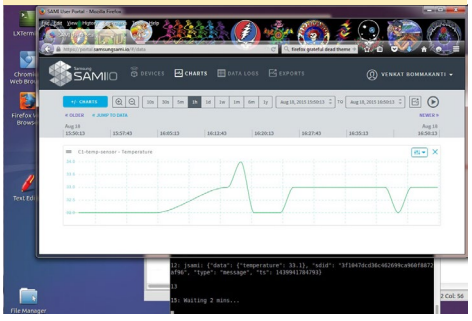
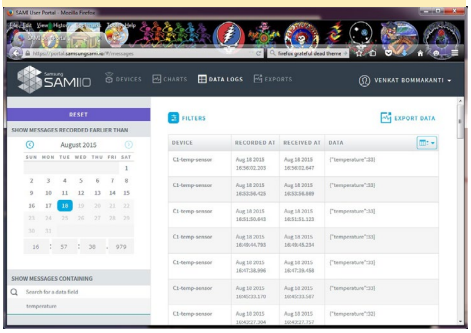


Figure 19: Temperature log on SAMIIO website

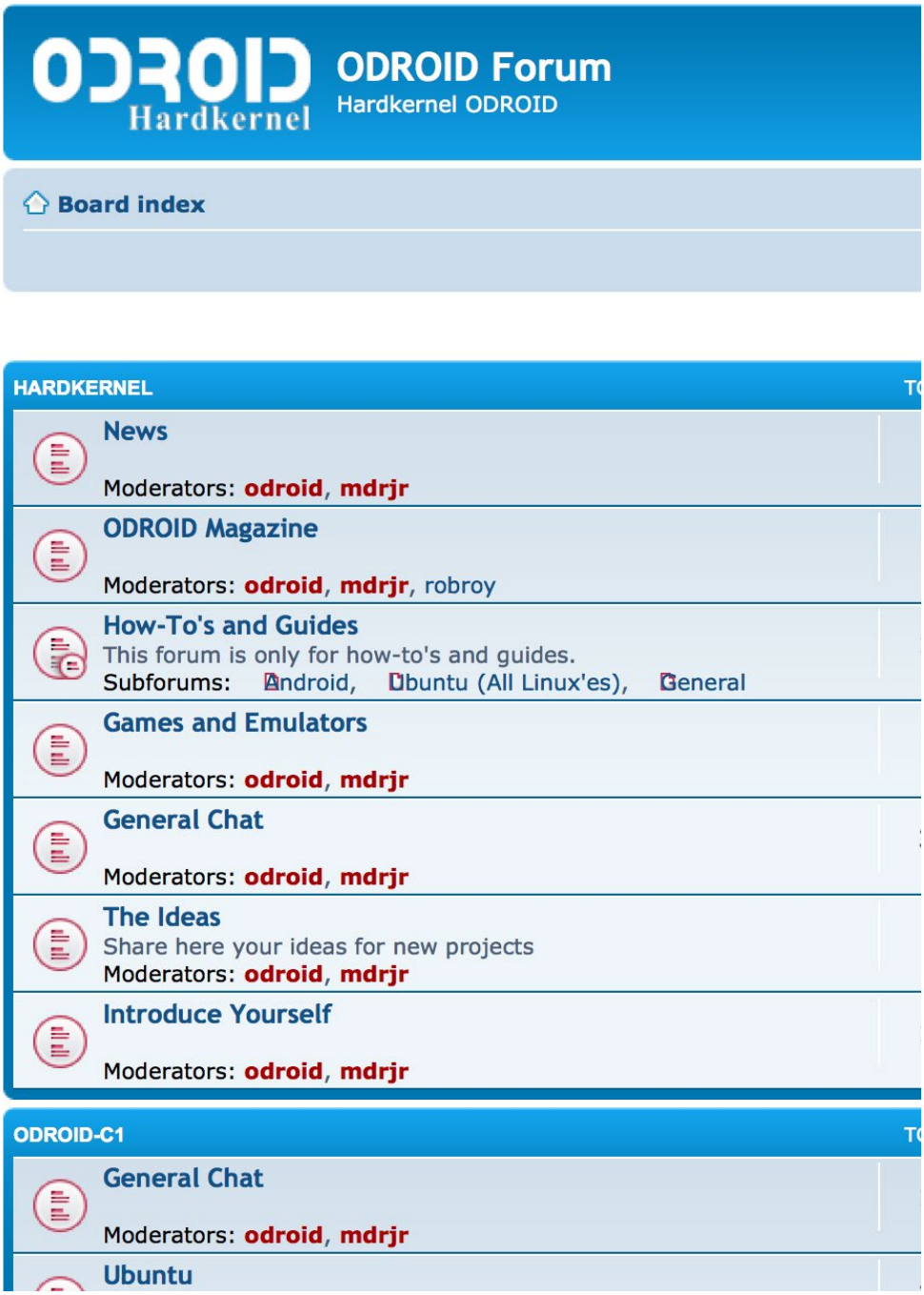


ODROID FORUMS

THE PERFECT PLACE TO COMMUNICATE WITH HARDKERNEL DEVELOPERS

by Rob Roy

The ODROID forums have been the central meeting place for the growing Hardkernel community for several years, with over 12,500 members as of September 2015. You can discuss ODROIDs with Mauro, the lead Linux kernel developer, and Justin, the CEO of Hardkernel, along with a growing team of developers who donate their time to helping you get the most out of your ODROID. Check it out at http://forum.odroid.com!



MEET AN ODROIDIAN

ULI MIDDLEBERG: LINUX WIZARD AND DOCKER SPECIALIST

edited by Rob Roy

Please tell us a little about yourself.

I'm living in Hamburg, Germany together with my wife and my two children. I'm 45 and working for a German newspaper publisher.

How did you get started with computers?

My first computer was an Atari ST 1040, where I learned programming (Pascal, MODULA 2, C, M86k Assembler). I was lucky to work with an Acron Archimedes A3000, which was light years ahead of everything else that I had access to at that time. At university, I was really happy to find a bunch of brand new NeXT Cubes and NeXT Stations that the computer science department has bought recently.

What attracted you to the ODROID platform?

The ODROID-C1 wasn't the first ARM device I've tried. When compared to the Raspberry Pi, it gives much better performance. Compared to the imx6 based devices, you'll experience a vivid user community and a very responsive support. The Hardkernel developers are open minded to contributions coming from the community, which makes it easy to integrate custom projects like Docker into the mainstream kernel. And I'm curious to see what ODROID device will be announced next.

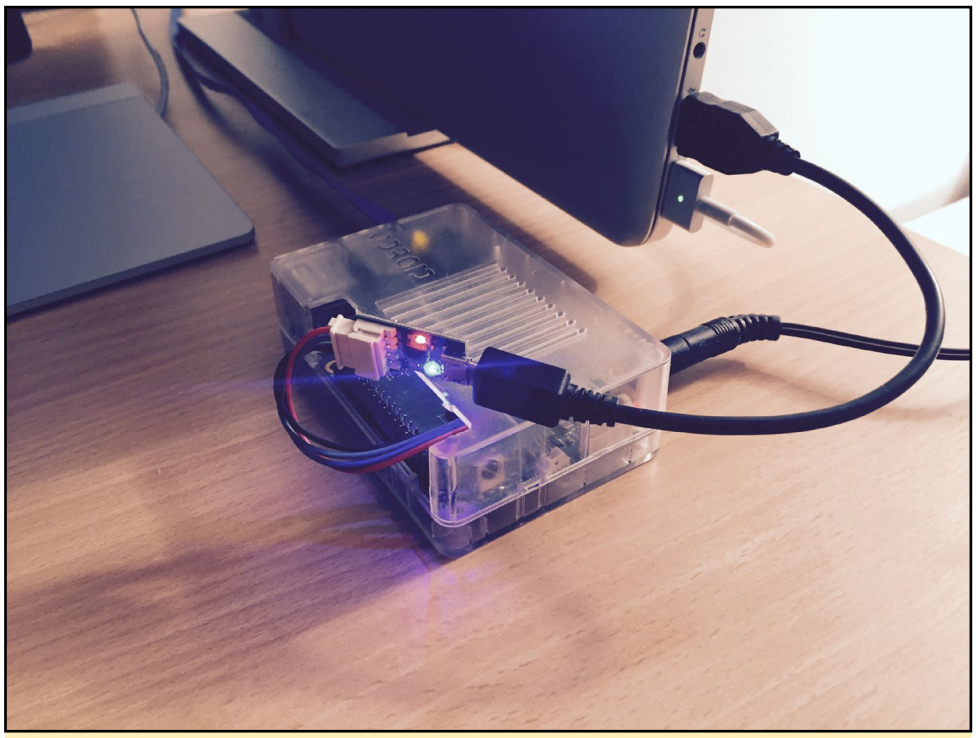
Uli regularly contributes his Docker expertise to the forums

How do you use your ODROIDs?

Currently I have a C1 only. I'm using it for developing, and for things like home automation using openHAB. I want to use it as a private syncing device together with OwnCloud in the future, but first I need to enable cryptodev support for offloading TLS/AES. I'm about to buy a C1+ and the audio board as a replacement for my CD player as well as a XU4 for developing purposes. Tools like Docker help you to move workloads between different devices easily, so maybe I'll move the OwnCloud service to the XU4, once I've managed to get Docker up and running there.

You are very knowledgeable about Linux, especially kernels. How did you become so proficient?

I started to use Linux in 1994, and two years later I worked as a system administrator at our university research group and started system administration by reading the System Administration book at http://oreil.ly/1Y1yWhH. At that time, compiling your own Linux kernel was a usual task. I continued to work as an administrator for a couple of years, but today I run Linux for my private purposes only. On the PC platform, there is almost no need to compile a custom Linux kernel anymore, since the distribution main-



This simplistic setup belies the powerful software running on Uli's trusty ODROID-CI

tainers/vendors do the job for you. When I started to work with ARM based Linux devices in 2013, I remembered the time when I had my first lessons in Linux. Today there are a lot of helpful resources available online, and it's very likely that someone else has raised your specific question before.

Which ODROID is your favorite?

I haven't worked with the U3 which most people would name as their favourite ODROID board. The XU4 has a very interesting price/performance ratio, but I'd love to see a 64-bit ARMv8 ODROID board.

Are you involved with any other computer projects unrelated to the ODROID?

I discovered Docker as a very fascinating way to develop, deploy and operate applications, but saw a lack of support for ARM based devices. So I started to port Docker on ARM, along with several others, and I wrote down all the necessary steps to do this in my GitHub wiki at http://bit.ly/1M5Iphp. This gives other interested people the chance to repeat these steps with their own boards, which I prefer in favour of issuing and maintaining custom images.

I really like the idea of running Linux on this small ARM boxes due to the power consumption vs performance ratio. Especially in small environments, an ARM based Linux Server will offer more than enough performance. So, I will continue writing down interesting things about Linux on ARM, just in case I need to remember how I did this particular thing.

What hobbies and interests do you have aside from computers?

Sports! I'm really keen on doing windsurfing, like my whole family. I like riding my bicycle instead of using a car, and I love to go jogging.

What type of hardware innovations would you like to see for future Hardkernel boards?

I'm certainly not the first one asking for this, but a fanless ARMv8/AArch64 board supporting more than 2 GB of RAM, SATA port or independent USB3 ports and mainline Linux kernel hardware support would be really cool.

What advice do you have for someone want to learn more about programming and/or hardware?

Start reading good books, set-up up your own Linux box, learn Python or Java, create your own project by modifying and extending existing ones, and get familiar with GitHub. There are also a lot of excellent MOOCs (http://bit.ly/1EX2vLx) covering a vast variety of different topics.

Uli enjoys windsurfing and other water sports with his children