LVM Data Migration • XU4 Fan Control • OSX USB-UART interfacing

ODROID Magazine

Year Two
Issue #22
Oct 2015

Apache TOMCAT

Your web server and servlet container running on the world's most power-efficient computing platform

Plex Media Server

Linux Gaming: Emulate Sega's last console, the Dreamcast

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

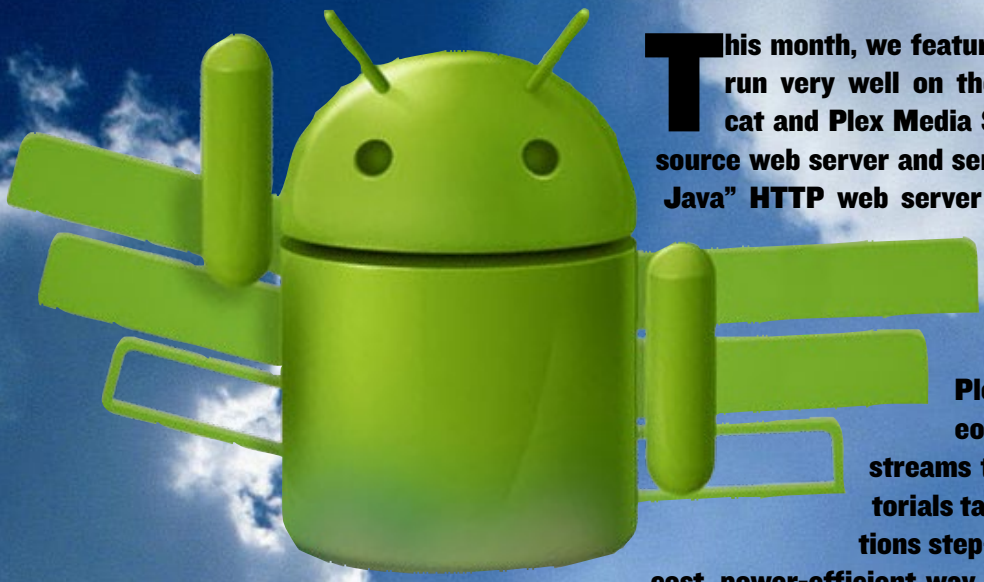
We are now shipping the ODROID-U3 device to EU countries! Come and visit our online store to shop!

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



This month, we feature two extremely useful servers that run very well on the ODROID platform: Apache Tomcat and Plex Media Server. Apache Tomcat is an open-source web server and servlet container that provides a "pure Java" HTTP web server environment for Java code to run in. It allows you to write complex web applications in Java without needing to learn a specific server language such as .NET or PHP. Plex Media Server organizes your video, music, and photo collections and streams them to all of your screens. Our tutorials take you through these server installations step-by-step so that you can enjoy a low-cost, power-efficient way to run an advanced server at home. The recent release of Lakka for the ODROID, an OpenElec-based distribution, makes it easier to play your favorite games. Tobias reviews the Dreamcast emulator, which is one of the most advanced console emulators available for the ODROID, Nanik continues to show us how to build Android for the ODROID-C1, Bruno details data migration using LVM, and we learn how to control the ODROID-SHOW using Python. As usual, we also present many beloved Linux game ports that will provide hours of fun!

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

The fall season on the northern hemisphere means that the summer is approaching Brazil, and this is the time when our fans kick high to keep our processors cool over here. Not that my ODROIDS suffer much though. Still, unfortunately for them, I'm the only one popping some cold beers at the pool over here.

Or maybe someday I'll devise a submarine beer drinking robotic ODROID...

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.

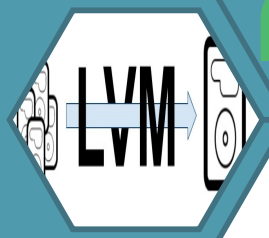
James LeFevour, Art Editor

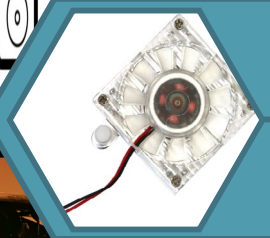
I am a Digital Media Specialist who is also enjoying freelance work in social network marketing and website administration. The more I learn about ODROID capabilities, the more excited I am to try new things I'm learning about. Being a transplant to San Diego from the Midwest, I am still quite enamored 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.

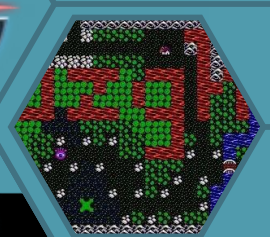
Manuel Adamuz, Spanish Editor

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



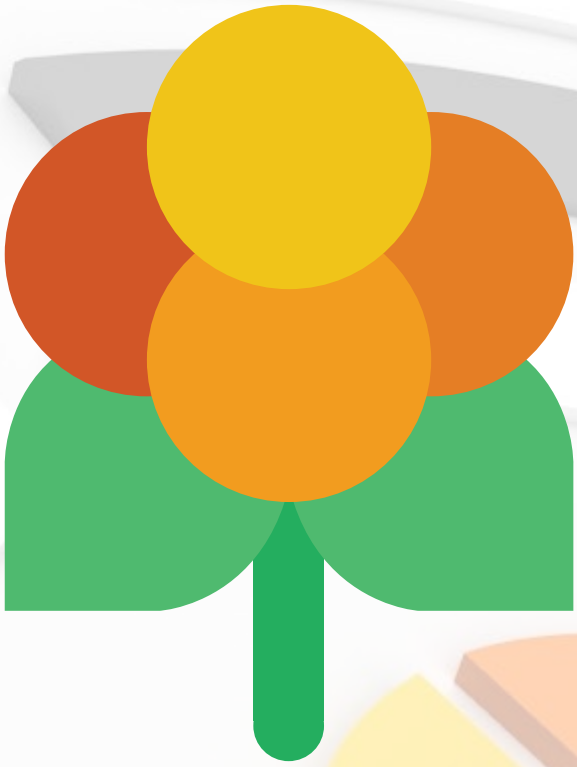




OS SPOTLIGHT: LAKKA ON THE ODROID-C1

DIY RETRO EMULATION CONSOLE

edited by Rob Roy



There are some great community gaming images available for the ODROID platform such as the Debian-based ODROID GameStation Turbo and the Android-based Pocket Rocket. The latest gaming image to be released is an open-source, multi-platform operating system based on OpenElec called Lakka, which uses the populator RetroArch software to provide console emulation for many different types of games. It has recently been ported to the ODROID-C1, is intended to be easy to setup and use, and supports the following systems:

- Atari 2600
- Atari Jaguar
- Atari Lynx
- Cave Story
- Dinothawr
- Doom
- FB Alpha
- FFmpeg
- Game Boy
- Game Boy Advance
- Game Boy Color
- Master System
- Mega Drive
- Nintendo Entertainment System (NES)
- Neo Geo Pocket
- PCEngine
- PlayStation
- PlayStation Portable (PSP)
- Sega 32X
- Super Nintendo Entertainment System (SNES)
- Vectrex

One of the benefits of Lakka is that it auto-detects many different types of pre-configured controllers, including Xbox 360, PS3/PS4, Saitek, Logitech, and Zeemote.

Getting started

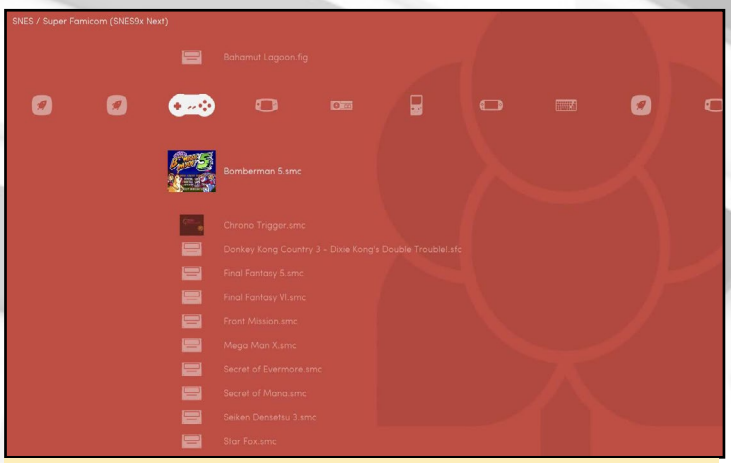
To install Lakka, download the pre-built image for the ODROID-C1 from `http://bit.ly/1YIOrvw` onto a host Linux system. Unzip the file, then determine the device name for the SD card by listing the current drives and partitions:

```
$ ls -l /dev/sd*

brw-rw---- 1 root disk  8,  0 22 mars  23:01 /dev/sda
brw-rw---- 1 root disk  8,  1 22 mars  23:01 /dev/sda1
brw-rw---- 1 root disk  8,  2 22 mars  23:01 /dev/sda2
brw-rw---- 1 root disk  8,  3 22 mars  23:01 /dev/sda3
brw-rw---- 1 root disk  8,  4 22 mars  23:01 /dev/sda4
brw-rw---- 1 root disk  8,  5 22 mars  23:01 /dev/sda5
brw-rw-r-- 1 root users 8, 16 22 mars  23:01 /dev/sdb
```

Those ending with numbers are partitions, and others are drives. In this example, sda is the main hard drive, and sda1 to sda5 are its partitions. Insert a blank microSD card into the host computer, and type the command again:

```
$ ls -l /dev/sd*

brw-rw---- 1 root disk  8,  0 22 mars  23:01 /dev/sda
brw-rw---- 1 root disk  8,  1 22 mars  23:01 /dev/sda1
brw-rw---- 1 root disk  8,  2 22 mars  23:01 /dev/sda2
brw-rw---- 1 root disk  8,  3 22 mars  23:01 /dev/sda3
brw-rw---- 1 root disk  8,  4 22 mars  23:01 /dev/sda4
brw-rw---- 1 root disk  8,  5 22 mars  23:01 /dev/sda5
brw-rw-r-- 1 root users 8, 16 22 mars  23:49 /dev/sdb
brw-rw---- 1 root disk  8, 17 22 mars  23:49 /dev/sdb1
brw-rw---- 1 root disk  8, 18 22 mars  23:49 /dev/sdb2
```

Notice that sdb is now filled with one or more partitions, which are shown as sdb1 and sdb2 in this example. This means that sdb represents the SD card reader, but it could be a different letter on your system. Make sure to adapt the rest of this tutorial to use your drive letter.



Lakka menu

Flash the image

Now that you know your SD card drive, navigate to the directory where you extracted Lakka, and flash the card, substituting your drive letter for sdX:

```
$ sudo dd if=Lakka-*.img of=/dev/sdX
```

It should take a few minutes until the prompt returns. Once it has completed, you can unplug your SD card and proceed to the next step.

First boot

To run Lakka, follow these steps:

Insert the microSD card into the ODROID-CI
Plug an HDMI cable between your ODROID and your TV
Turn on the TV
Plug in the ethernet cable to the ODROID-CI (optional)
Plug one of the supported joypads into one of the 4 ODROID's USB ports
Plug in the power supply of the ODROID-CI

You should see the Lakka splash screen, as shown in Figure 1. The package will then automatically expand the filesystem and reboot after about 30 seconds. This happens only on the first boot, and subsequent boots should be much faster. If everything went well, you should now be able to navigate Lakka Menu, our graphical interface, as shown in Figure 2. Congratulations, you have successfully installed Lakka!

Playing games

Insert a USB drive containing the ROMs that you'd like to use. Your USB drive must be formatted as FAT or NTFS. The partition will be mounted automatically in a new folder under /storage/roms/, and your ROMs will appear in the Lakka menu.

Manufacturer	Model	BIOS	Additional info
3DO	3DO (4DO)	panafz10.bin	
Atari	Atari 2600 (Stella)	'not needed'	
Atari	Atari 7800 (ProSystem)	ProSystem.dat	Atari 7800 Database
Atari	Atari 7800 (ProSystem)	7800 BIOS (U).rom	Atari 7800 BIOS
Atari	Lynx (Handy)	lynxboot.img	(Lynx Boot Image)
Id Software	Doom (PrBoom)	prboom.wad	(PrBoom WAD) Need to be near your Doom wads
Magnavox	Odyssey2	o2rom.bin	(Odyssey 2 BIOS)
NEC	PC-FX	pcfx.bios	
NEC	PC Engine/PCE-CD	syscard3.pce	(PCE-CD BIOS)
Nintendo	NES (Nestopia)	disksys.rom	(Famicom Disk System BIOS)
Nintendo	N64 (Mupen64Plus)	'not needed'	
Nintendo	Game Boy Advance	gba_bios.bin	(GBA BIOS)
Sony	PS (Beetle PSX)	scph5500.bin	(PS1 JP BIOS)
Sony	PS (Beetle PSX)	scph5501.bin	(PS1 US BIOS)
Sony	PS (Beetle PSX)	scph5502.bin	(PS1 EU BIOS)
Sega	MegaCD	bios_CD_E.bin	(MegaCD EU BIOS)
Sega	SegaCD	bios_CD_U.bin	(SegaCD US BIOS)
Sega	MegaCD	bios_CD_J.bin	(MegaCD JP BIOS)
Sega	Saturn	saturn_bios.bin	(Saturn BIOS)
SNK	NeoGeo	neogeo.zip	(NeoGeo BIOS) MUST BE PLACED IN ROMS FOLDER

Lakka BIOS table

Some libretro cores require a BIOS to work. You need to find those BIOSes by yourself as it is illegal to provide them. Those BIOSes must be placed in the "system" folder on your Lakka Box. Figure 3 outlines the different BIOS files that are required to emulate each type of system. Remember that Linux is a case-sensitive system, so it will be necessary to rename the BIOS files according to this table, so Lakka will be able to find them. Figure 4 details the file extensions used for the various emulators.

For more information, or to post comments, questions or suggestions about Lakka, please visit the Lakka home page at `http://www.lakka.tv`, or the Libretro forums at `http://bit.ly/1P09vcs`.

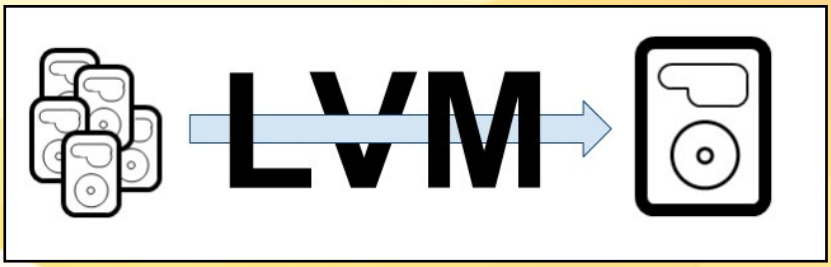
System	Model	Filename Extension	Additional info
3DO	3DO (4DO)	.iso .cue	
Atari	Atari ST/STE/TT/Falcon	.st .msa .zip	
Atari	Atari 2600	.a26 .bin	
Atari	Atari 7800 (ProSystem)	.a78 .bin	
Atari	Lynx (Handy)	.lnx	
Id Software	Quake	.pak	
MSX	MSX (fMSX)	.rom .mx1 .mx2	
Magnavox	Odyssey2	.bin	
NEC	PC-FX	.cue .ccd .toc	
NEC	PC Engine/PCE-CD	.pce .cue .ccd	
Nintendo	SNES / Super Famicom	.sfc .smc .bml	
Nintendo	NES / Famicom (FCEUmm)	.fds .nes .unif	
Nintendo	N64	.n64 .v64 .z64 .bin .u1	
Nintendo	Game Boy Advance	.gba .agb .bin	
PSX	PS (Beetle PSX)	.bin .cue .toc .m3u .ccd .exe .mdf .cbn .pbp	
Sega	MS/GG/MD/CD +GX	.mdx .md .smd .gen .bin .cue .iso .sms .gg .sg	
Sega	Sega Saturn	.bin .cue .iso	
Sega	Sega Saturn	.bin .cue .iso	

ROM extension table

LOGICAL VOLUME MANAGEMENT

MAKE YOUR DATA MIGRATION EASIER WITH LVM

by David Gabriel



Let's say that you just bought a brand new ODROID-XU4 and want to migrate all of your running services from an old ODROID to this new one. You already have it all configured for your needs, and having to set up everything again by creating logical volumes, file systems, and setting permissions would take a long time. There is also the time spent transferring all files to the new ODROID that, depending on the amount of files, could take a lot of time.

Having all of your files configured under an external drive using LVM can make all this migration process a lot easier and take just a few minutes. How? By exporting your volume group from the old system and importing it back on the new one.

Please note that all commands are run with root privileges. First, you will need to stop all services running under that volume group, which usually done by typing the following:

```
# service <daemon name> stop
```

Then, unmount all the file systems attached to volume group. In my case, I only have /home on it.

```
# umount /home
```

Next, you have to deactivate the logical volumes (LVs) under the volume group that you are going to export. You should check current status first:

```
# lvscan
```

The above command will give you the status of the LVs, showing them under /dev/<vg_name>. This is the same as the /dev/mapper/<vg_name>-<lv_name> structure that we saw in the previous article. They are both links to the actual lvm block file on /dev/dm-x. To deactivate the logical volume, type:

```
# lvchange -a n \
  /dev/rootvg/homelv
```

If you have more than one, just paste them one after the other, separated by spaces. If you run lvscan again, you will see that the LVs changes from active to inactive. Once all of the LVs from the VG are inactive, you can export the VG:

```
# vgexport rootvg
```

You can then do a final check by running vgscan, and it will show you that the volume group is now exported. At this point, you can remove your drive from your old system and plug it onto the new one. Then, you can run pvscan, and you should see all your partitions showing on the new system.

To import your data, type:

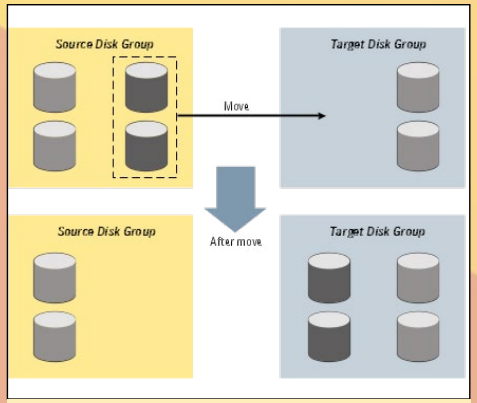
```
# vgimport rootvg
```

This should allow you to see your logical volumes. Now, just mount the file system back:

```
# mount /dev/rootvg/homelv
```

This should restore everything to the new ODROID without wasting time copying or recreating all of the structures that you already had. Of course, you still have to reinstall your software in the likely case that they were not stored on the LVM.

Now you know how to migrate your data between different systems by taking advantage of the features of LVM. I hope this helps you save time when you want to move your files to another computer.



Moving volumes across disk groups is easy

ODROID-XU4 FAN CONTROL

THE COOL WAY TO MANAGE YOUR XU4 TEMPERATURE AND POWER USAGE

by @Grotus

The ODROID fan driver uses pulse width modulation (PWM) to control the speed of the fan, with the PWM duty cycle adjusted based on the temperature of the CPU. The driver has four speed settings, which it selects among based on three temperature settings. So, if the current CPU temp is below the lowest temperature setting it uses the first fan speed, when between the first and second temperature settings it uses the second fan speed, when between the second and third temperature settings it uses the third fan speed, and when greater than the third temperature setting it uses the fourth fan speed.

There are a number of settings in sysfs for the ODROID-fan driver, and on the XU4 this is under /sys/devices/ODROID_fan.13, while on the XU3 it is /sys/devices/ODROID_fan.14.

The settings are:

fan_mode: Auto or Manual (set to 1 for auto and 0 for manual, default Auto)
fan_speeds: Four space delimited fan speed percentage values, in ascending order. (default "1 51 71 91")
pwm_duty: Current setting of the PWM duty cycle (0-255, set dynamically)
pwm_enable: On or off (default on)
temp_levels: Three space delimited CPU Celsius temperature values, in ascending order (default "57 63 68")

There are two places to get the CPU temperature in sysfs: /sys/devices/10060000.tmu/temp and /sys/devices/virtual/thermal/thermal_zone0/temp. The first is readable only by root or a user in group root, and shows the temperatures of five sensors. The second is readable by all and gives a single temperature. In both cases, the temperatures are in milli-degrees Celsius, so you'll need to divide by 1000 in order to get Celsius. The highest of the values in /sys/devices/10060000.tmu/temp is used to control the fan speed.

The fan speed settings are specified as percentages, and should be in the range 0-100. The PWM duty cycle is specified in the range 0-255, and in automatic mode is calculated by multiplying the fan speed setting by 255 and dividing by 100. For example, in the default case, when the temperature hits 57 degrees it turns the fan on to 51%, which equals a PWM duty cycle of 51*255/100 = 130.

The fan-control script works by setting the fan_mode to manual and changing the pwm_duty to the desired value based on the temperature. The script has 9 fan levels defined as opposed to the 4 in the ODROID-fan driver. In order to configure the automatic mode for the fan, you can echo new settings to the fan_speeds and temp_levels settings, which will take effect immediately.

Example

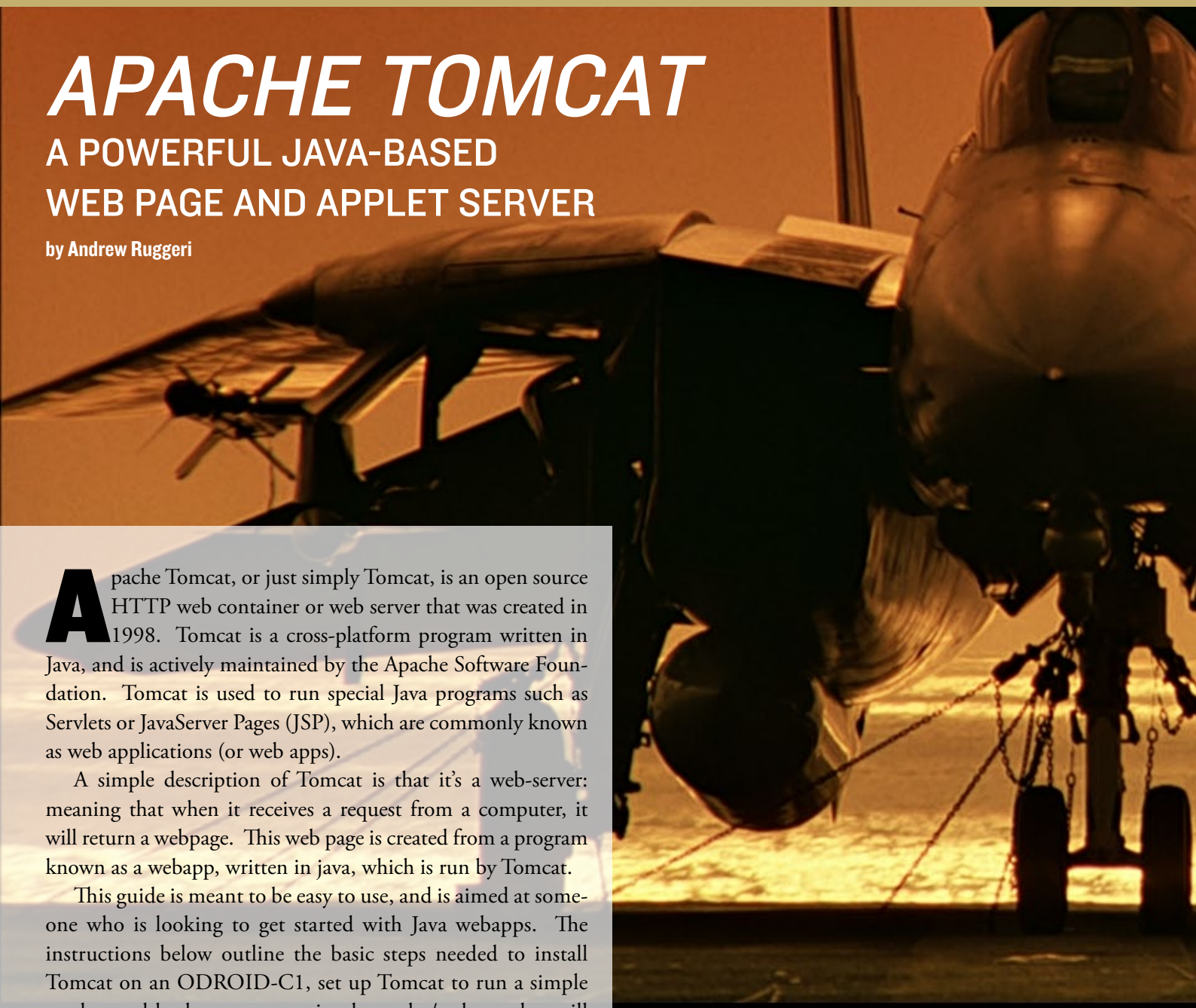
Here is an example to make the fan turn on to 20% at 50C, go up to 50% at 70C, and up to 95% at 80C on the ODROID-XU4:

```
$ sudo echo "1 20 50 95" > /sys/devices/ODROID_
fan.13/fan_speeds
$ sudo echo "50 70 80" > /sys/devices/ODROID_fan.13/
temp_levels
```

Setting the values in this way will not persist over a reboot. To have the settings applied at boot time, you can set a rule for udev, by creating a file in /etc/udev/rules.d with the desired settings. I used 60-ODROID_fan.rules as the name on my system. The following should work on either an XU3 or XU4 as it matches based on the driver name of ODROID-fan rather than the kernel name which is different in the two versions:

```
DRIVER=="ODROID-fan", ACTION=="add", ATTR{fan_
speeds}="1 20 50 95", ATTR{temp_levels}="50 70 80"
```

To post comments, questions or suggestions, please visit the original thread at `http://bit.ly/1jit0Rx`.



APACHE TOMCAT

A POWERFUL JAVA-BASED WEB PAGE AND APPLET SERVER

by Andrew Ruggeri

Apache Tomcat, or just simply Tomcat, is an open source HTTP web container or web server that was created in 1998. Tomcat is a cross-platform program written in Java, and is actively maintained by the Apache Software Foundation. Tomcat is used to run special Java programs such as Servlets or JavaServer Pages (JSP), which are commonly known as web applications (or web apps).

A simple description of Tomcat is that it's a web-server: meaning that when it receives a request from a computer, it will return a webpage. This web page is created from a program known as a webapp, written in java, which is run by Tomcat.

This guide is meant to be easy to use, and is aimed at someone who is looking to get started with Java webapps. The instructions below outline the basic steps needed to install Tomcat on an ODROID-C1, set up Tomcat to run a simple servlet, and lastly to create a simple servlet/webapp that will post the C1's CPU temperature. Although this guide is written for the ODROID-C1, similar steps would be needed for other devices.

Installing Tomcat

There are several ways to install Tomcat onto the C1, the three most popular are via source compile and install, 'apt-get'ting, and having it run as stand-alone. A quick apt-cache search for 'Tomcat' shows that Tomcat 7 is available in the C1's default repos. For the sake of simplicity, we will do an install from apt-get using the following commands:

```
sudo apt-get install tomcat7
```

[Optional] documentation can be downloaded with:

```
sudo apt-get install tomcat7-docs
```

[Optional] various Tomcat examples can be downloaded with:

```
sudo apt-get install tomcat7-examples
```

While the newest version of Tomcat is 8, it is only available through other repositories or by building the source. To keep this guide at a beginners level, I will focus only on Tomcat 7 installed from the default repositories. If you do wish to install Tomcat 8 from source, you can still follow this guide as the steps past installation are unchanged.

Running & Testing

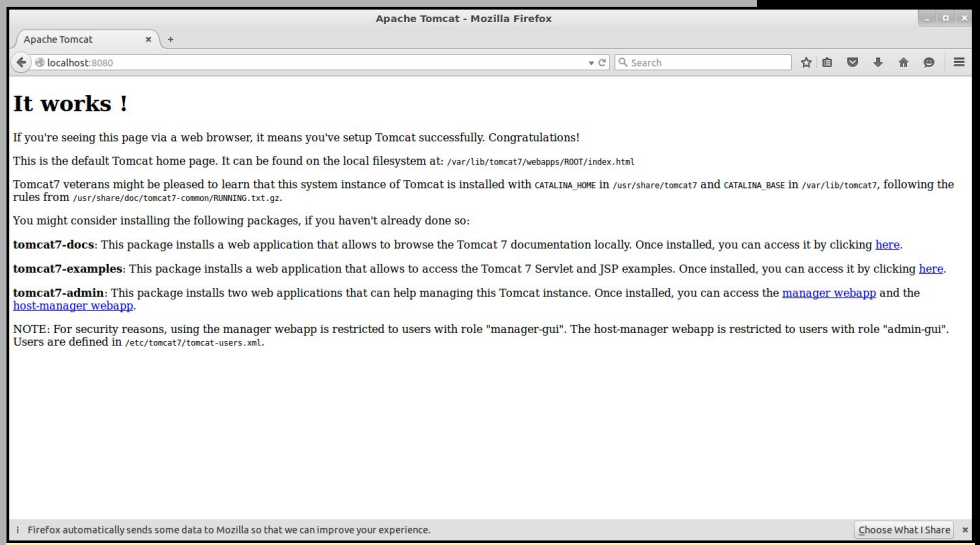
Once installed, Tomcat will run as a service and can be turned on, restarted, or stopped with the following commands.

```
sudo service tomcat7 start
sudo service tomcat7 restart
sudo service tomcat7 stop
```

Now the simplest way to test if Tomcat7 is up and running properly is to see if Tomcat's test web page will load. Open a web browser (Firefox, Chromium, ect) on the C1. In the navigation text box type:

```
localhost:8080
```

Lets look at what the address means: firstly, localhost is telling the browser to look at the local computer that it's running on (likewise when you type google.com, you're telling the browser to look for the computer that google.com is running on). '8080' is the port Tomcat is receiving connections on (8080 is the default). This value can be changed, and is discussed in the optional configuration section. If everything is working correctly, the following web page should load.

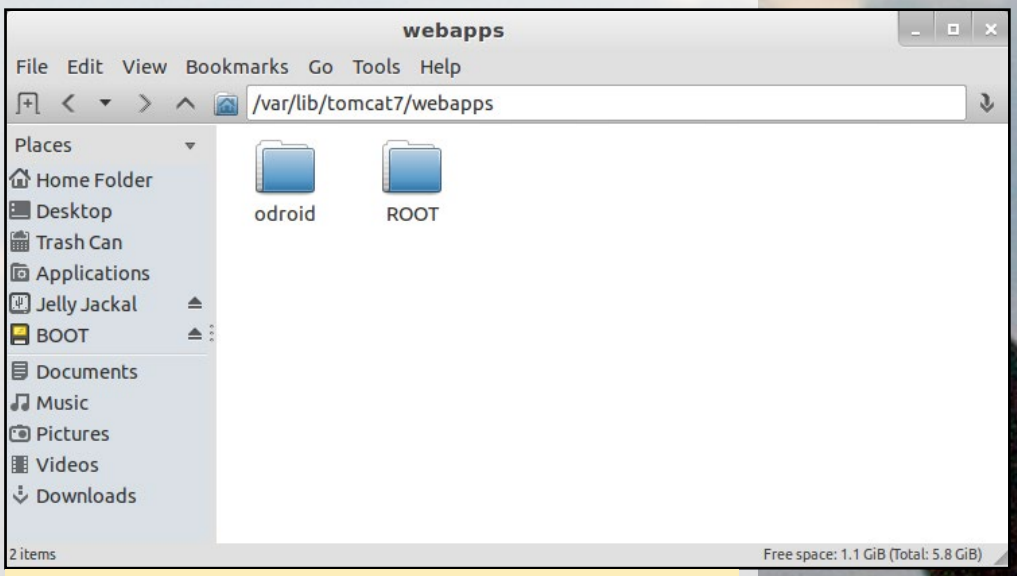


Whenever you install Tomcat you will see this page confirming that it is running OK.

Configuring [Non-Optional]

Following an apt-get install, we need to configure a few things to get Tomcat up and running with our servlet which we are going to make. What we are going to be doing is telling Tomcat what to do when it receives a web request, or HTTP request as we will be using.

Navigate to /var/lib/tomcat7/webapps/ in either a file browser or terminal. There should be 1 folder in there already: 'ROOT', and this is the default webpage we saw during testing. We are now going to set up a place for our new page. Start by creating a folder called "odroid" (you might need to be root to do so) in the webapps directory, so you should now have both a ROOT and an odroid folder side-by-side.



Here at webapps is where your webpages are going to be placed.

Now go into the odroid folder (/var/lib/tomcat7/webapps/odroid/) and create a folder named "WEB-INF". Once again, move into the newly create "WEB-INF" folder and create a folder named "classes". You should now have created a total of 3 folders (marked in bold) with the following paths:

```
/var/lib/tomcat7/webapps/odroid/
/var/lib/tomcat7/webapps/odroid/WEB-INF/
/var/lib/tomcat7/webapps/odroid/WEB-INF/classes/
```

odroid: This folder, and any other folder in the "webapps" folder (such as ROOT), are known as a "document base directory". This is the folder where any assistant files to the webapp, such as images, javascript files, CSSs, or additional HTML files, should be placed.

WEB-INF: Every "document base directory" contains this folder. Inside each one of these folders, you will see a 'classes' directory as well as a file named "web.xml". The "web.xml" files

will be discussed in more detail below.

classes: This is the folder which will contain the compiled java servlet files.

Now that we have all the directories in place, navigate to the WEB-INF folder /varr/lib/tomcat7/odroid/WEB-INF/. Open up a file editor (gEdit, kate, nano, vim, etc.) and create a new xml file named "web.xml". This is the file that will tell Tomcat which servlet to run when it receives a web request. The content of the web.xml file is as follows:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app version="3.0" xmlns="http://java.sun.com/
xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance" xsi:schemaLocation="http://java.
sun.com/xml/ns/javaee http://java.sun.com/xml/ns/ja-
vaee/web-app_3_0.xsd">

    <servlet>
        <servlet-name>odroidTemperature</servlet-
name>
        <servlet-class>temperatureServlet</servlet-
class>
    </servlet>

    <servlet-mapping>
        <servlet-name>odroidTemperature</servlet-
name>
        <url-pattern>/temperature</url-pattern>
    </servlet-mapping>

</web-app>
```

The very top of this xml document is standard, you will see those same lines in every web.xml file, and they are need to set-up the xml namespaces. The two parts in the middle are what we will focus on as they are specific to each servlet web page we set-up. The servlet element always needs to go before the servlet-mapping element. The servlet elements describe the servlet that will produce the html, and servlet-mapping describes the url path that will call this servlet. It can be thought of as a two part system. If you look inside "servlet" and "servlet-mapping" you'll notice they both contain 'servlet-name' and have the same value for 'odroidTemperature'. This is because the servlet-name links both the 'servlet' element and the 'servlet-mapping' element together. Up in the servlet element there is something called 'servlet-class', this is simply the name of the servlet we are going to create (we will create a file later on called "temperatureServlet.java"). Lastly there is "url-patern" found in the servlet-mapping element. This will currently tell Tomcat to wait for any url with /temperature at the end of it, and

if found, will send it to the servlet named temperatureServlet.

Configuring [Optional]

Tomcat is highly configurable, however it will still work straight from an install. If you wish to customize Tomcat more this is done by editing a few xml files. The additional xml files are located in /etc/tomcat7 and these files are context.xml, server.xml, web.xml. To edit these files you can use any text editor of your choice (vim, gedit, nano, etc.). Several of these files contain many parameters that can be changed. Below is just a quick overview of each of these xml files. Take a look at the xml file itself, or check out Apache's documentation for more in-depth info.

server.xml : Changes Tomcat itself. While making the xml changes listed below, have a look at the other possible changes which could be made as well. This is the file in which you can change the Tomcat's default port from 8080. Note that for debug it is advisable to use ports above 1024.

context.xml : Changes the behavior of Tomcat. One change to this file that you might wish to make is to have Tomcat automatically refresh a web page on a code change. This is very useful during debug, but should be turned off during normal use as it adds unneeded overhead.

web.xml : The properties in this xml are the default properties used for all web applications.

Creating a servlet

What is a java servlet? A simple definition of a servlet is that it is a java program that takes in information sent to it by a web browser, and answers with HTML. For this example we are going to make a servlet call odroidTemp (easier than odroid_Temperature). When it receives a request from a browser, it will create a web page that will display the temperature of the odroid.

To get started with our servlet we will create a java file directly in the 'classes' folder. Open up a text editor and create a new file with the name "temperatureServlet.java". When you save the file, make sure it's saved to the location: /var/lib/tomcat7/webapps/odroid/WEB-INF/classes/.

```
import java.io.*;
// From /usr/share/tomcat7/lib/servlet-api.jar
import javax.servlet.*;
import javax.servlet.http.*;

public class temperatureServlet extends HttpServlet {

    @Override
    public void doGet(HttpServletRequest request,
```

```
HttpServletResponse response) throws IOException,
ServletException {

        // MIME type
        response.setContentType("text/html");

        PrintWriter htmlResponse = response.getWrit-
er();
        try {
            // Get the Tempurature
            String TempuratureValue;
            BufferedReader br = new
BufferedReader(new FileReader("/sys/devices/virtual/
thermal/thermal_zone0/temp"));
            // 1 line file with current temp
            TempuratureValue = br.readLine();
            // Clean it up a bit
            TempuratureValue = TempuratureValue ==
null ? "NA" : TempuratureValue.substring(0,2);

            // HTML TIME
            // Open
            htmlResponse.println("<html>");
            htmlResponse.println("<head><title>Odroid
Tempurature</title></head>");
            htmlResponse.println("<body>");

            // Show Tempurature
            htmlResponse.println("<h1>Odroid Tempura-
ture</h1>");
            htmlResponse.println("<p>Tempurature C: "
+ TempuratureValue + "</p>");

            // Close
            htmlResponse.println("</body>");
            htmlResponse.println("</html>");

        } finally {
            // Close the writer and we're finished
            htmlResponse.close();
        }
    }
}
```

You should have a basic understanding of java, and can get the gist of the functionally that is happening in the java code above. If not, don't panic, there are MANY great beginner level guides to java. Without getting too caught up in the basics of java, I would like to explain at a higher level what is happening in servlet. This servlet overrides the doPost function, which is what intercepts a HTTP POST request (likewise there is a get-

Post which intercepts an HTTP GET request). The response that this servlet returns is a string of HTML that is formed in the try block, and set by the MIME type.

The next step is to compile the code to be used by Tomcat, and we will do that straight from the terminal. Open the terminal, go to the directory where you have the temperatureServlet.java file, and run the following commands.

```
$ javac -target 1.7 -source 1.7 -cp .:/usr/share/\
tomcat7/lib/servlet-api.jar temperatureServlet.java
```

The javac command invokes the java compiler which will take our java source code and compile it into a program that Tomcat can run. We add the 'target' 1.7 and 'source' 1.7 arguments to tell the compiler to compile for java 1.7. This is done because Tomcat7 will run with JVM 1.7, but invoking javac alone will compile for Java 1.8 which will cause backwards compatibility problems. The third part of the command you will notice is "-cp .:/usr/share/tomcat7/lib/servlet-api.jar" which makes the java compiler use the servlet-api.jar to help build the temperatureServlet.java program. We need to add this because we are, as you know, making a java servlet that needs some help from that external servlet-api.jar file.

Running the Servlet

Before we drive right in and check the servlet, first do a quick check of the odroid webapp file structure (a quick check can save you from a big headache). Your folder structure should match the structure below:

```
Webapps\
      odroid\
              WEB-INF\
                      web.xml
                      classes\
                       temperatureServlet.java
                       temperatureServlet.class
```

Alright! Now it's time to test and see if everything works. If you have not yet done so, restart Tomcat with the "sudo service tomcat7 restart" command. Now open up a browser and load the page "localhost:8080\odroid\temperature". If all went well, the following page should have loaded.

Congratulations! that's everything for this guide. As you can imagine, there is much, much more information on this topic, and I would highly recommend that you use this guide as a starting point and use the examples and guides from Apache: see `tomcat.apache.org/tomcat-7.0-doc/index.html` for further information.

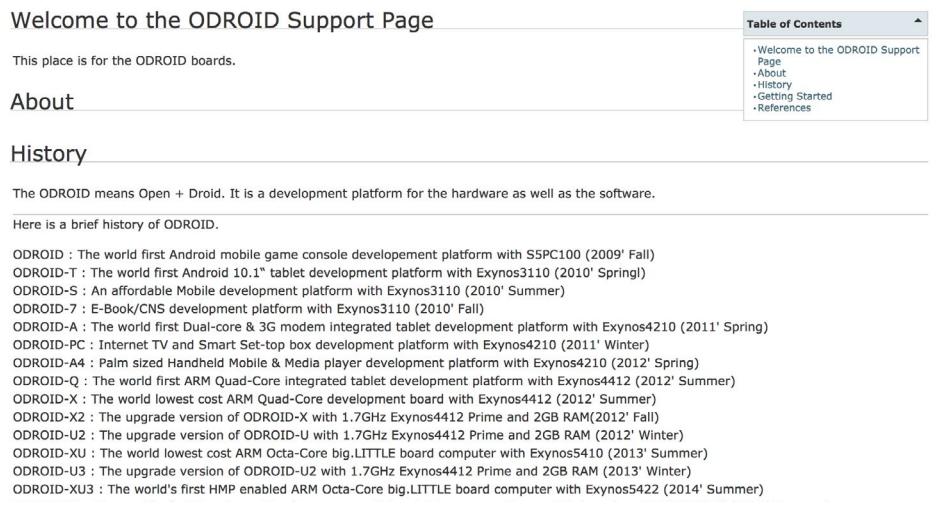
COMMUNITY WIKI

CONTRIBUTE TO THE EXPANDING ODROID KNOWLEDGE BASE

by Rob Roy

Hardkernel has recently set up a great resource for ODROIDians to contribute their knowledge to a community wiki, available at `http://wiki.odroid.in`. It is intended to complement the official Hardkernel wiki at `http://bit.ly/1R6DOgZ`, and is useful for posting your tips, community image links, projects, and anything else that might be beneficial to the Hardkernel community.

If you'd like to participate, click on the "Request Account" button in the top right, and include your ODROID forum username in the "Personal Biography" section. For comments, questions and suggestions related to the new wiki, please visit the original forum thread at `http://bit.ly/1QDMNoT`.



ADRENALINE-CHARGED FUN

SPEEDY NINJA, THE NEW ENDLESS RUNNER YOU WERE LOOKING FOR

by Bruno Doiche

Here at the magazine design office, alongside my trusty ODROID cluster stack, a good combination of beer and coffee and lots of articles, I enjoy testing every single endless runner that appears in front of me. Among the dozens that I have played, Speedy Ninja certainly is worth of your attention. Demanding as much reflexes as the ability to think, you have to be always aware of what is happening from both sides in order to continue collecting as many coins as you can. The reward? A superfun dragon ride while being a ninja!

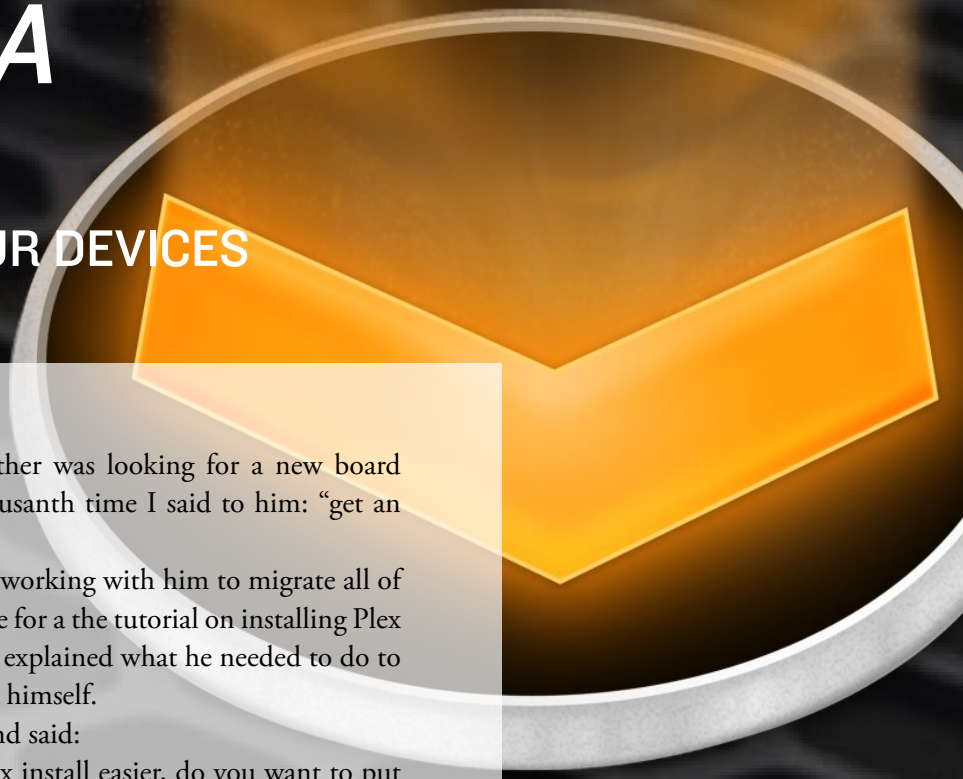
`https://play.google.com/store/apps/details?id=com.netease_na.nmd2`

For every move and every feat achieved, an amazing dragon ride awaits!

PLEX MEDIA SERVER

YOUR MEDIA ON ALL YOUR DEVICES

by Bruno Doiche and Rick Doiche



About two months ago, my younger brother was looking for a new board to use as a media server, and for the thousanth time I said to him: "get an ODROID!".

I then gave him an ODROID-XU4 and was working with him to migrate all of his content to his new machine, when he asked me for a the tutorial on installing Plex Media Server. I pointed to the Plex website and explained what he needed to do to get it installed, and he said that he could do it by himself.

A few days later, he contacted me on a chat and said:

"Guess what, I wrote a script to make the Plex install easier, do you want to put in the magazine?"

So, without further ado, here is my brother's script for installing Plex Media Server on your ODROID:

```
#!/bin/bash


###################################
# Install Plex Media Server
#
#
# Odroid Magazine 2015
# http://magazine.odroid.com/
#
# This script will install
# Plex Media Server
# Beta 0.2
# 26 Aug 2015
###################################


# Cheching system packages dependencies
DEPENDENCIES(){
PACK_LIBC="libc6-armel";
PACK_MULTILIB="gcc-multilib";
CHECK=$(dpkg-query -l $PACK_LIBC $PACK_MULTILIB > /dev/null 2>&1 ; echo
$?);
if [ "$CHECK" -eq "1" ]; then
      echo "Installing Packages $PACK_LIBC and $PACK_MULTILIB "
      apt-get install -y libc6-armel gcc-multilib ;
      locale-gen en_US.UTF-8 ;
```

```
      dpkg-reconfigure locales ;
      else
      echo "INFO: Packages $PACK_LIBC and $PACK_MULTILIB are installed
already"
fi }

# Creating a build in environment
BUILD(){

URL="https://downloads.plex.tv/plex-media-serv-
er/0.9.12.11.1406-8403350/";
PLEX_SPK="PlexMediaServer-0.9.12.11.1406-8403350-arm.spk";

mkdir /tmp/plex ; cd /tmp/plex ;
wget -P /tmp/plex $URL$PLEX_SPK ;
mv $PLEX_SPK PlexMediaServer.tgz ;
tar -xvf PlexMediaServer.tgz ;
mkdir /tmp/plex/package ;
tar -xvf /tmp/plex/package.tgz -C /tmp/plex/package ;
mkdir -p /apps/plexmediaserver/Binaries ;
mv /tmp/plex/package/** /apps/plexmediaserver/Binaries ;
mkdir /apps/plexmediaserver/temp ;
mkdir /apps/plexmediaserver/MediaLibrary ; };
touch /var/log/plex/plexms.log ; chown plex /var/log/plex/plexms.log ;


ADD_PLEX(){

K=$(useradd plex -s /bin/bash -d /home/plex ; echo $?);
if [ "$K" -eq "0" ]; then
      mkdir /home/plex
      chown -R plex.plex /home/plex
      echo "INFO: Plex user has been created sucessfully";
      else
      echo "INFO: Plex user already exists";
      fi };


REMOVE_TEMP_BUILD(){
rm -rf /tmp/plex ; };


UNISTALL(){
/etc/init.d/plex stop
userdel plex ;
rm -rf /home/plex /apps/plexmediaserver /etc/default/plexmediaserver_en-
vironment /etc/init/plexmediaserver.conf  /etc/plex /etc/init.d/plex /
var/log/plexms.log ;
update-rc.d plex remove ;
};
```

```
PLEX_CONF(){
mkdir /etc/plex ;
ln -s /home/plex/Library/Application\ Support/Plex\ Media\ Server/Prefer-
ences.xml /etc/plex/Preferences.xml ;
};

CALL_INFO(){
echo -e "\033[01;31m# ODROID MAGAZINE - Plex installation script
\033[00;37m"
echo -e "\033[01;31m# INFO:\033[00;37m";
echo -e "\033[01;31m# ------------------------------------------------
-------------------------\033[00;37m"
echo -e "\033[01;31m# Plex script will install Plex media Server on your
system.\033[00;37m";
echo -e "\033[01;31m# As requirement this script must be run as
root.\033[00;37m"
echo -e "\033[01;31m# Plex script will also add \"plex\" user to your
system in order to avoid security issues\033[00;37m";
echo -e "\033[01;31m# Directories as /home/plex and /apps/plexmediaserver
will be created.\033[00;37m";
echo -e "\033[01;31m# Plex script requires internet access once it has to
access and download Plex media server from http://Plex.tv\033[00;37m";
echo -e "\033[01;31m# Please note that some System package libs are also
required and script will try to install it\033[00;37m";
echo -e "\033[01;31m# Libs: libc6-armel and gcc-multilib \033[00;37m"
echo -e "";
echo -e "";
echo -e "TERM:"
echo -e "\033[01;31m# Running this script you acknowledge and accept that
ODROID MAGAZINE will not be responsible for any damage caused in your
system. \033[00;37m"
echo "";

echo -e "\033[01;31m# ------------------------------------------------
-------------------------\033[00;37m" ;
echo " ";
echo " ";
echo -e "\033[01;32mOps.. please try $0 {install|uninstall|info}\033[00;
37m";
echo -e "\033[01;32mExample: $0 install\033[00;37m";
echo "";
echo -e "May the force be with you";

};

DEBIAN_SYSTEM_SCRIPT(){


sudo bash -c "cat <<EOT > /etc/init.d/plex
#!/bin/bash
```

```
##     ##  ###   ##  ##    ##  ###   ###
##     ##  ## #  ##  ##    ##      ##
##     ##  ##  # ##  ##    ##     ##
#####  ##  ##   ###  ########  ###   ###

########################################
# Henrique Doiche                      #
# Plex Media Center                    #
# http://www.plexapp.com/            #
# Last edition 07-02-2015              #
# Plex script                          #
########################################
### BEGIN INIT INFO
# Provides:          scriptname
# Required-Start:    \\\$remote_fs \\\$syslog
# Required-Stop:     \\\$remote_fs \\\$syslog
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: Start daemon at boot time
# Description:       Enable service provided by daemon.
### END INIT INFO

#############Comment#####################
# NOTE:
# You can also add plex script into Debian /etc/init.d/
# and add it to run in startup as priority 50
# update-rc.d plex defaults 50
# That way you are \"Debian\" compliance
#
#
# Old school
# If you add this script named as plex in
# /usr/local/bin directory, Linux will
# be able to read it as System default \\\$PATH
# so you'll be able to add it into rc.local:
# Add: \"plex start\" to your /etc/rc.local
# file. without "" =)
# Ex: Script Location: /usr/local/bin/plex
# rc.local exemple: cat /etc/rc.local
# plex start


########################################

START(){
        sudo su - plex -c \"/apps/plexmediaserver/Binaries/start.sh > /
var/log/plex/plexms.log 2>&1 &\" ;
        echo -e \"Starting Plex [\033[01;32m Done \033[00;37m]\"; }

STOP(){
```

```
        PIDS=\\\$(ps aux | grep plex | grep -v grep | grep -v root |  awk
{'print \\\$2'});
        PIDS_DLNA=\\\$(ps aux | grep DLNA | grep -v grep | awk {'print
\\\$2'});
        if [ -z \"\\\${PIDS}\" ] || [ -z \"\\\${PIDS_DLNA}\" ]; then
                echo \"Plex isn't running. Nothing to do.\";
                else
                echo  \"Starting graceful shutdown...\";
                kill -s TERM $PIDS $PIDS_DLNA 2> /dev/null ;
                sleep 5;
                if [ -z \"\\\$PIDS\" ] && [ -z \"\\\$PIDS_DLNA\" ] ; then
                        echo -e \"Graceful shutdown was [\033[01;32m Suc-
cessful \033[00;37m] \";
                        else
                        echo -e \"Plex process are still running. Killing
Process [\033[01;32m Done \033[00;37m]\";
                        kill -9 \\\$PIDS \\\$PIDS_DLNA 2> /dev/null ;
                fi
        fi }



RESTART(){
        STOP;START;
        echo -e \"Restarting Plex [\033[01;32m Done \033[00;37m]\"; }

STATUS(){
        STATUS=\\\$(ps aux | grep plex | grep -v grep | grep -v root |
awk {'print \\\$2'});
        if [ -z \"\\\$STATUS\" ]; then
                echo \"Plex isn't running\";
                else
                echo -e \"Plex is running on PIDs \n\033[01;31m\\\$STATUS
\033[00;37m\";
        fi
}

case \\\$1 in
'start') START ;;
'stop') STOP ;;
'restart') STOP; START ;;
'status') STATUS ;;
*)
echo \"Ops.. please try \\\$0 {start|stop|restart|status}\";
exit 0
;;
esac



EOT"
```

```
chmod 755 /etc/init.d/plex

}


PLEX_MEDIA_CONF(){
bash -c "cat <<EOT > /etc/init/plexmediaserver.conf
# plexpms - service job file

description \"Plex Media Server\"
author \"http://www.plexapp.com/\"

# When to start the service
start on runlevel [2345]

# When to stop the service
stop on runlevel [016]

# Automatically restart process if crashed
respawn

# Sets nice and ionice level for job
nice -5

# What to execute
script
/etc/init.d/plex
end script

EOT"
};


PLEX_MEDIA_ENV(){
# Creating plexmediaserver_environment

bash -c "cat <<EOT > /etc/default/plexmediaserver_environment
# default script for Plex Media Server

# the number of plugins that can run at the same time
PLEX_MEDIA_SERVER_MAX_PLUGIN_PROCS=6

# ulimit -s \\\$PLEX_MEDIA_SERVER_MAX_STACK_SIZE
PLEX_MEDIA_SERVER_MAX_STACK_SIZE=3000

# uncomment to set it to something else
PLEX_MEDIA_SERVER_APPLICATION_SUPPORT_DIR=\"/apps/plexmediaserver/Medi-
aLibrary\"

# let's set the tmp dir to something useful.
TMPDIR=\"/apps/plexmediaserver/temp\"
```

```
# We need to catch our libraries
LD_LIBRARY_PATH=\"/apps/plexmediaserver/Binaries:\\\$LD_LIBRARY_PATH\"

EOT"
};


PLEX_STARTUP(){
# Creating start.sh
rm -rf /apps/plexmediaserver/Binaries/start.sh ;
bash -c "cat <<EOT > /apps/plexmediaserver/Binaries/start.sh
#!/bin/bash

#SCRIPTPATH=\\\$(dirname \\\$(python -c 'import sys,os;print os.path.
realpath(sys.argv[1])' \\\$0))
SCRIPT=\\\$(readlink -f \\\$0)
SCRIPTPATH=\\\`dirname \\\${SCRIPT}\\\`
export LD_LIBRARY_PATH=\"\\\${SCRIPTPATH}\"
export PLEX_MEDIA_SERVER_HOME=\"\\\${SCRIPTPATH}\"
export PLEX_MEDIA_SERVER_MAX_PLUGIN_PROCS=6
export LC_ALL=\"en_US.UTF-8\"
export LANG=\"en_US.UTF-8\"
ulimit -s 3000
cd \\\${SCRIPTPATH}
./Plex\ Media\ Server
EOT"
chmod 755 /apps/plexmediaserver/Binaries/start.sh ;
};


ROOT=$(whoami);

case $1 in
'install')

if [ "$ROOT" == "root" ]; then
      clear ;
      DEPENDENCIES ;
      ADD_PLEX ;
      BUILD ;
      PLEX_CONF ;
      PLEX_MEDIA_ENV ;
      PLEX_MEDIA_CONF ;
      PLEX_STARTUP ;
      DEBIAN_SYSTEM_SCRIPT ;
      REMOVE_TEMP_BUILD ;
#     update-rc.d plex defaults;
      clear ;
      echo "-----------------------------------------------";
      echo "INFO:";
```

```
      echo "Please use service plex start | service plex stop | service
plex restart";
      echo "Plex installation completed";
      echo "You can reach server typing http://127.0.0.1:32400/web/index.
html into browser";
      echo "Install completed";
      else
      echo -e "\033[01;31mINFO:\033[00;37m";
      echo -e "\033[01;31mPlex installation script must be run as root
user\033[00;37m";
fi

 ;;
'uninstall')

UNISTALL ;;
'info')

CALL_INFO ; ;;


*)
CALL_INFO ;
exit 0
;;
esac
```

This script has been tested to work with the U2, U3, X2, XU3, and XU4 models. It is also available for download at http://bit.ly/1LgYazS.

In case you are not using the default Linux distribution provided by Hardkernel, you will need to create a directory at /var/log/plex and give 777 permission to it using chmod.

I ended asking Rick if he wanted to write more articles covering the things about which he is a Linux expert, but he just said, "Meh, not right now bro, someday... who knows!"

Bruno still hopes his brother writes another magazine article with him

USING THE USB-UART WITH MAC OSX

HELPING OUR MAC USERS GET CONSOLE ACCESS TO THEIR ODROIDS

by @midel

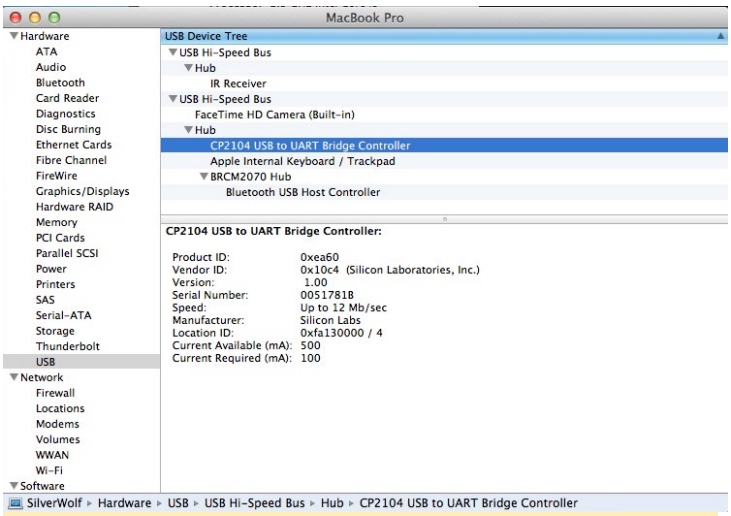
If you own a Macintosh as your primary computer, you can use Hardkernel's USB-UART kit to read console output from an ODROID without needing to install a virtual Linux machine, since OSX is based on BSD. This article details how to install the necessary drivers and software in order to use a Macintosh as a debugging console for ODROID development.

OSX can be used to connect to your ODROID's console

Driver installation

The first step is to obtain the drivers for the UART, which may be downloaded from `http://bit.ly/1Fk1rBu`. Unzip and install the package, then reboot the system.

Make sure that the driver is installed correctly by plugging in the UART into the USB port of your Mac, and checking system information under **Apple Logo > About This Mac > More Info... > System Report... > Hardware > USB**, then looking for the CP2104 USB to UART Bridge Connector.



Checking system information to verify driver installation

Software setup

Minicom allows the console output from the ODROID to be displayed on the Macintosh screen. In order to install Minicom, it's necessary to first install homebrew from `http://bit.ly/1R4sYYX`, as well as to install the Command Line Tools (CLT) for Xcode from `http://apple.co/1JsNXyi`. The CLT package is required for building software with brew, ports, or fink. Since you are a developer on the Mac Platform, it's a good idea to pick it up if you want to be able to use the standard GNU Linux tools on your machine.

Open the Terminal application, found in /Applications/Utilities/Terminal. Install Minicom with this command:

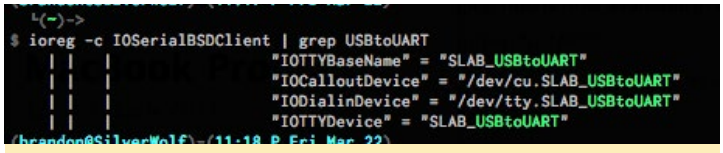
```
$ brew install minicom
```

It will take a few moments to compile the program. The next step is to get the actual terminal device name for the UART KIT.

Connecting the UART

Now that we have the drivers and Minicom installed, it's time to find out where it is. Enter the following command into a Terminal window. The output will look similar to Figure 3.

```
$ ioreg -c IOSerialBSDClient | grep USBtoUART
```

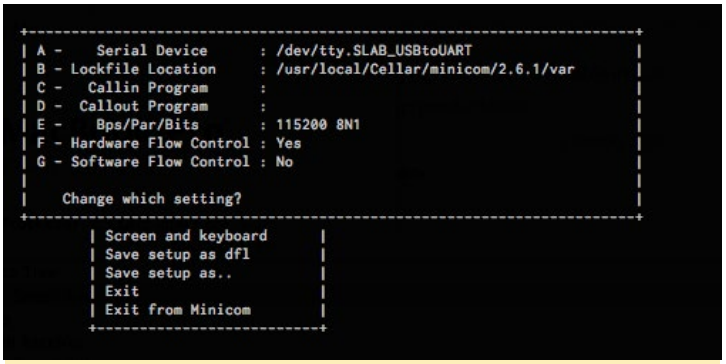


ioreg output

Next, open Minicom in SETUP mode with this command:

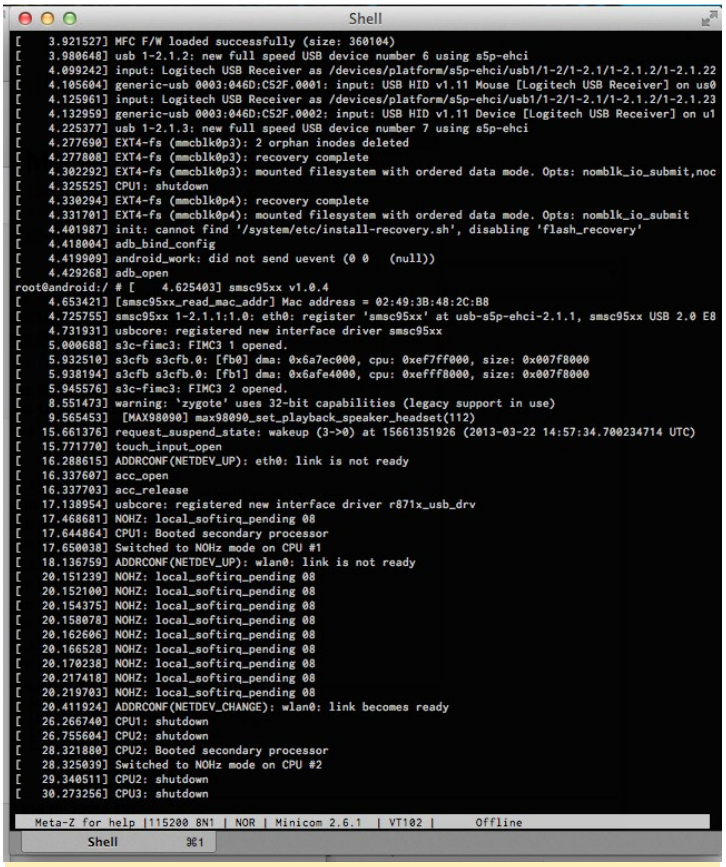
```
$ sudo minicom -s
```

Enter your user password, press enter, then navigate with the arrow keys to Serial port setup and type A to change the Serial Device to what we got from the previous command. The hardware flow control should be OFF, and the software flow control should be ON.



Changing the serial device using Minicom

Press enter until you return to the main menu, and navigate to "Save setup as dfl" in order to save the configuration so that we never need to set it up again. Then select Exit to enter the main view, then exit Minicom completely using Esc+X.



Sample output from USB-UART device

Logging

Type the following at the command prompt to start Minicom, then press ESC+L.

```
$ sudo minicom
```

Provide a filename and press enter. Run your Odroid and capture your input, then stop or close the logging with ESC+L again. You will find the log file in whatever was the current directory when Minicom was started.

Tips

If you don't want to use MiniCom, OSX comes with GNU Screen as part of the default install. This means that once you've installed the USB-UART drivers, you can find the correct port to connect to with the following command:

```
$ ls -l /dev/tty.*
```

On my system, the UART device is listed as "dev/tty.SLAB_USBtoUART". You can then connect to the ODROID using the "screen" application:

```
$ screen "dev/tty.SLAB_USBtoUART 115200
```

To post comments, questions, or suggestions regarding using the USB-UART kit with OSX, please visit the original thread at `http://bit.ly/1Wm6BRs`.

This article finally gives us a reason to actually use a Macintosh

ANDROID DEVELOPMENT

BUILDING ANDROID FOR THE ODROID-C1 - PART 2

by Nanik Tolaram

In my previous article, I discussed how to build Android for the ODROID-C1, and hopefully by now you are familiar with building Android images from scratch, and have done some experimentation with the board. In this article, I will discuss the Android boot process for the C1, since the booting process is slightly different than what is normally found on other ODROID boards.

selfinstall-odroidc.bin

The ODROID-C1 build system outputs one single file called selfinstall-odroidc.bin that needs to be copied to the microSD card. What is fascinating about this file is that it contains all of the relevant Android images ready for use with the board, as shown in Figure 1.

This file acts as a container, hosting varieties of file that are

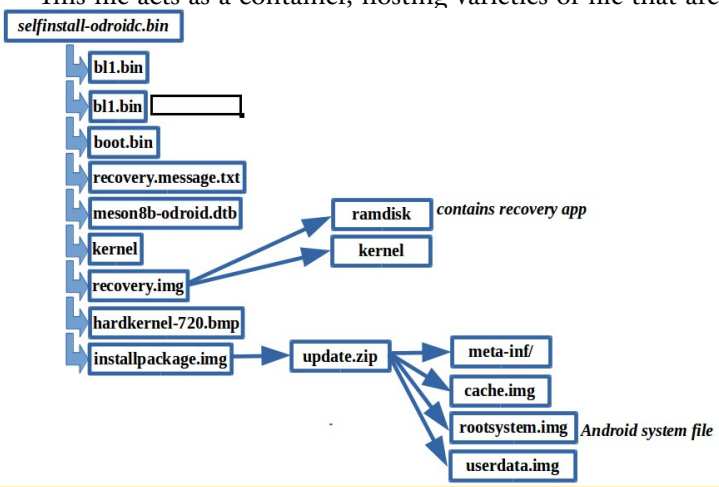


Figure 1: Content of selfinstall-odroidc.bin

bundled together, which form a complete package for installation process. The magic of extracting and installing these different filesystems during boot process is done by the recovery application that is loaded inside the recovery.img file. Figure 2 shows the snippet of the build script that put the different files together forming one single .bin file.

```
$(PRODUCT_OUT)/selfinstall-$(TARGET_DEVICE).bin: \
    $(INSTALLED_BOOTIMAGE_TARGET) \
    $(INSTALLED_RECOVERYIMAGE_TARGET) \
    $(PRODUCT_OUT)/bl1.bin.hardkernel \
    $(PRODUCT_OUT)/u-boot.bin \
    $(PRODUCT_OUT)/installpackage.img
    @echo "Creating installable single image file..."
    dd if=$(PRODUCT_OUT)/bl1.bin.hardkernel of=$@ bs=1 count=442
    dd if=$(PRODUCT_OUT)/bl1.bin.hardkernel of=$@ bs=512 skip=1 seek=1
    dd if=$(PRODUCT_OUT)/u-boot.bin of=$@ bs=512 seek=64
    dd if=$(RECOVERY_MESSAGE_FILE) of=$@ bs=512 seek=1016
    dd if=$(PRODUCT_OUT)/meson8b_odroidc.dtb of=$@ bs=512 seek=1088
    dd if=$(PRODUCT_OUT)/kernel of=$@ bs=512 seek=1216
    dd if=$(INSTALLED_RECOVERYIMAGE_TARGET) of=$@ bs=512 seek=17600
    dd if=$(PRODUCT_OUT)/hardkernel-720.bmp of=$@ bs=512 seek=33984
    dd if=$(PRODUCT_OUT)/installpackage.img of=$@ bs=512 seek=49152
    sync
    @echo "Done."

.PHONY: selfinstall
selfinstall: $(PRODUCT_OUT)/selfinstall-$(TARGET_DEVICE).bin
```

Figure 2: Build script packaging selfinstall-odroidc1.bin

It is important to note that changing the sequence or removing anything from the build script can render your image unbootable. Changes in the build script require changes in other part of the bootloader.

Boot Flow

The boot process for this particular board is slightly more complicated than normal. There are 2 phases of the boot process, as shown in Figure 3. The 1st boot phase checks whether the microSD card has been formatted, and formats it if necessary. Once 1st boot phase has completed, it will continue to boot the board and move on to the 2nd boot phase by launching the Android init process.

When the board is powered on, the Amlogic chip will execute the first part of the boot process by running the bl1.bin main bootloader that is provided by the chip manufacturer. Upon completion of the main bootloader, the U-Boot will start executing, which is the part of the bootloader that decides what the next step will be, based on whether the microSD card has been formatted. When U-Boot finds that the microSD card has not been formatted, it will format the card with the

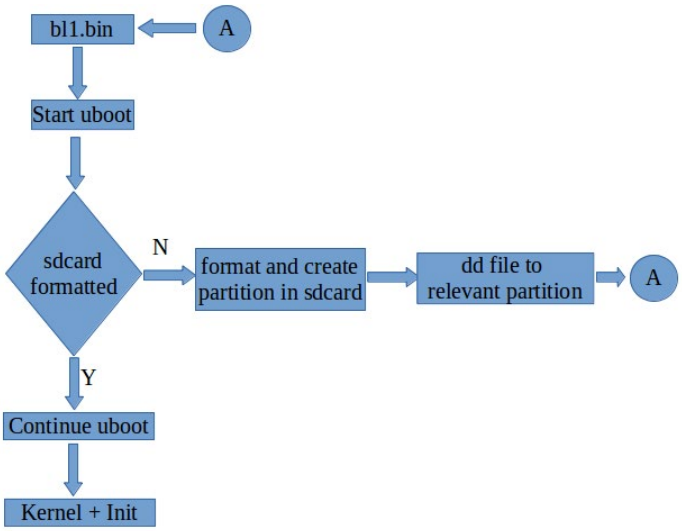


Figure 3: Boot flow ODROID-CI

relevant partition type and copy files to it. Once this process completes, it will then reboot the board. On the 2nd boot phase, if the U-Boot detects that the relevant partition exists, it will hand over control to the kernel.

U-Boot

Table 1 shows a partial log message from the U-Boot during the 1st boot phase of the bootup process, where it can be seen that it created the missing required Android partitions. Once it completes the formatting, it runs the recovery application and copies over the images to the newly created partition.

Since in the 1st boot phase there are no partitions on the microSD card, how does U-Boot know what to read, and from where? To find out, let's take a look at how the microSD card will look internally before any partitions are created by U-Boot. Figure 4 shows that the card contains different types of information such as the recovery application, bootloader, logo, kernel and much more. Obviously, when looking at this kind of information, U-Boot must have a similar structure stored somewhere in the source code that it can use to read the information, which is show in Figure 5.

Table I: Ist boot phase log

```
------------------------------------------------------------------------
* Welcome to Hardkernel's ODROID-C... (Built at 19:33:00 Dec  8 2014) *
------------------------------------------------------------------------
CPU : AMLogic S805
MEM : 1024MB (DDR3@792MHz)
BID : HKC1310001
….
….
U-boot-00000-gc878e20-dirty(odroidc@c878e205) (Jun 28 2015 - 20:20:12)
DRAM:  1 GiB
relocation Offset is: 2ff1c000
….
….
mmc save env ok
vpu clk_level = 3
….
….
============================================================
Vendor: Man 035344 Snr a1a3a066 Rev: 8.0 Prod: SU08G
            Type: Removable Hard Disk
            Capacity: 7580.0 MB = 7.4 GB (15523840 x 512)
------------------------------------------------------------
Partition      Start Sector      Num Sectors      Type
bad MBR sector signature 0x0000
============================================================

Creating default partition...
partion #     size(MB)      block start #     block count     partition_Id
    1           2924           9486336          5988352           0x0C
    2           1024             49152          2097152           0x83
    3           3072           2146304          6291456           0x83
    4            512           8437760          1048576           0x83
 MMC read: dev # 0, block # 17600, count 16384 ... 16384 blocks read: OK
## ANDROID Format IMAGE
## Booting kernel from Legacy Image at 12000000 ...
   Image Name:   Linux-3.10.33
   Image Type:   ARM Linux Kernel Image (lzo compressed)
   Data Size:    5008332 Bytes = 4.8 MiB
   Load Address: 00208000
   Entry Point:  00208000
```

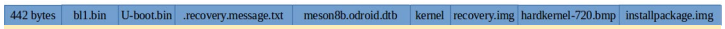


Figure 4: SDCard content on first phase boot

Utilizing the sys_partitions[] array, U-Boot is able to determine the location of the information that it needs to use. For example, it knows where the recovery application is located, so that it can read it from the microSD card and put in memory to be executed. Once all of the required files are in place, the boot process can continue and eventually display the desktop and become ready for user interaction.

```
struct fbt_partition fbt_partitions[] = {
        {
                .name = "system",               /* 2nd primary partition */
                .type = "ext4",
                .size_kb = BOARD_SYSTEMIMAGE_PARTITION_SIZE * 1024
        }, {
                .name = "userdata",             /* 2rd primary partition */
                .type = "ext4",
                .size_kb = BOARD_USERDATAIMAGE_PARTITION_SIZE * 1024
        }, {
                .name = "cache",                /* 3rd parimary partition */
                .type = "ext4",
                .size_kb = BOARD_CACHEIMAGE_PARTITION_SIZE * 1024
        }, {
                .name = "fat",                  /* 1st primary partition */
                .type = "vfat",
                /* FIXME: MUST fit in remaind blocks after followed by this */
                .size_kb = 1024 * 1024,
        }, {
                .name = 0,
                .type = 0,
                .size_kb = 0
        },
};
```

Figure 5: Location in sdcard for different content

Partition

Figure 6 shows the different partition created by U-Boot during the 1st boot phase. As can be seen in the figure, the different partition contains the /system, /data, /cache and vfat storage. U-Boot also stores all the partition information internally inside the source code, as seen in Figure 7.

The fbt_partitions[] structure contains the different partition that U-Boot will create during the 1st boot phase. You can cross reference and see that the partition size information outlined in size_kb field outlined in the structure matches with the log output from Table-1

Fancy Trick

To better understand the content of the selfinstall-odroidc.bin, I will show you how to extract the .bmp file that is used as the U-Boot logo. Remember the following step is only after you flash the selfinstall-odroidc.bin into your sdcard). To

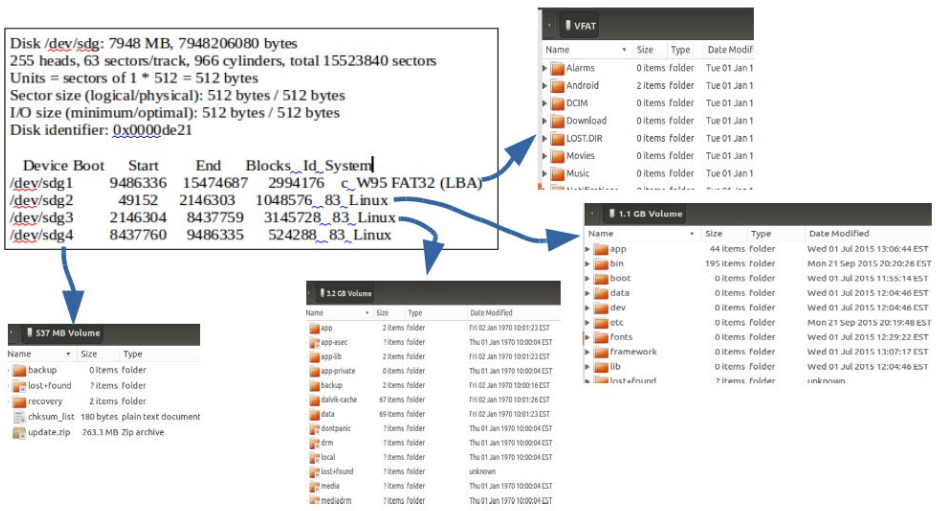


Figure 6: Different Android partitions

```
struct fbt_partition fbt_partitions[] = {
        {
                .name = "system",               /* 2nd primary partition */
                .type = "ext4",
                .size_kb = BOARD_SYSTEMIMAGE_PARTITION_SIZE * 1024
        }, {
                .name = "userdata",             /* 2rd primary partition */
                .type = "ext4",
                .size_kb = BOARD_USERDATAIMAGE_PARTITION_SIZE * 1024
        }, {
                .name = "cache",                /* 3rd parimary partition */
                .type = "ext4",
                .size_kb = BOARD_CACHEIMAGE_PARTITION_SIZE * 1024
        }, {
                .name = "fat",                  /* 1st primary partition */
                .type = "vfat",
                /* FIXME: MUST fit in remaind blocks after followed by this */
                .size_kb = 1024 * 1024,
        }, {
                .name = 0,
                .type = 0,
                .size_kb = 0
        },
};
```

Figure 7: U-Boot partition information

extract the .bmp from the SD card, type the following statement into a Terminal window:

```
$ sudo dd if=/dev/sdg \
  of=logo.bmp bs=512 \
  skip=33984 count=5400
```

You will see a file called logo.bmp in the current directory. Once you make modification to the .bmp file, you can put it back into the SD card by using the following statement:

```
$ sudo dd if=./logo.bmp \
  of=/dev/sdg bs=512 \
  seek=33984
```

FREE ORION

CONQUER THE GALAXY

by Tobias Schaaf

FreeOrion is a free, open source, turn-based space empire and galactic conquest computer game. It's inspired by the Master of Orion games. You can watch a gameplay video at http://bit.ly/1LD0x3R.

Prerequisites

First, update your kernel using the ODROID-Utility script. Then, link the Mali drivers (on the XU3 and XU4, use libmali.so instead of libMali.so):

```
$ sudo ln -sf /usr/lib/arm-linux-
gnueabihf/mali-egl/libMali.so \
/usr/lib/arm-linux-gnueabihf/\
libGLESv1_CM.so
$ sudo ln -sf /usr/lib/arm-linux-
gnueabihf/mali-egl/libMali.so \
/usr/lib/arm-linux-gnueabihf/\
libGLESv2.so
$ sudo ln -sf /usr/lib/arm-linux-
gnueabihf/mali-egl/libMali.so \
/usr/lib/arm-linux-gnueabihf/\
libEGL.so
$ cd ~ && mkdir freeorion
$ wget http://oph.mdrjr.net/\
meveric/other/freeorion/\
libgl-odroid_20150922-1_armhf.deb
$ wget http://oph.mdrjr.net/\
meveric/other/freeorion/\
libglues-odroid_\
20140903-1_armhf.deb
$ wget http://oph.mdrjr.net/\
meveric/other/freeorion/\
libglew-odroid_1.11.0-2_armhf.deb
```

Ubuntu 14.04

```
$ wget http://oph.mdrjr.net/\
meveric/other/freeorion/\
```

```
freeorion-data_0.4.5-\
1~ppa1~trusty1_all.deb
$ wget http://oph.mdrjr.net/\
meveric/other/freeorion/\
freeorion_0.4.5-1~\
ppa1~trusty1_armhf.deb
```

Ubuntu 15.04

```
$ wget http://oph.mdrjr.net/\
meveric/other/freeorion/\
freeorion-data_0.4.5-\
1~ppa1~vivid1_all.deb
$ wget http://oph.mdrjr.net/\
meveric/other/freeorion/\
freeorion_0.4.5-1~\
ppa1~vivid1_armhf.deb
```

Debian Jessie

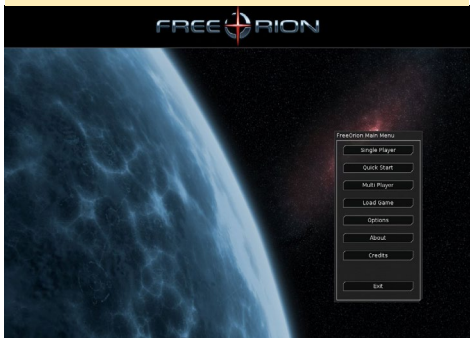
```
$ wget http://oph.mdrjr.net/
meveric/pool/main/f/freeorion-
odroid/freeorion-odroid_0.4.5-
1+deb8_armhf.deb
```

Installation

```
$ sudo apt-get install gdebi
$ sudo gdebi libgl-*.deb
$ sudo gdebi libglues-*.deb
$ sudo gdebi libglew-*.deb
$ sudo gdebi freeorion-data*.deb
$ sudo gdebi freeorion_*.deb
```

To play, click on the FreeOrion icon in the Games section of the Applications menu. Note that if you're using the GameStation Turbo image, the only step required is to type "apt-get install freeorion-odroid". For comments, suggestions, and questions, please visit the original thread at http://bit.ly/1OwEb6i, or check out the FreeOrion beginner's guide at http://bit.ly/1KULQsv.

FreeOrion has gorgeous graphics



HAXIMA NAZGHUL

A NEW ADVENTURE FOR ULTIMA V FANS

by @petevine

Haxima Nazghul is a CRPG (Computer Role Playing Game) that is modeled after the popular Ultima series. It is a top-down adventure fantasy game that provides a separate story line from the original Ultima game, while providing a similar visual and gameplay experience. It is specifically modeled after Ultima V, so if you've played that game, then Haxima Nazghul should feel very familiar.

Installation

Download the source code and game data from http://bit.ly/1MOCvEE, then unpack it by typing the following into a Terminal window:

```
$ cd ~/Downloads
$ tar xvzf nazghul-0.7.1.tar.gz
```

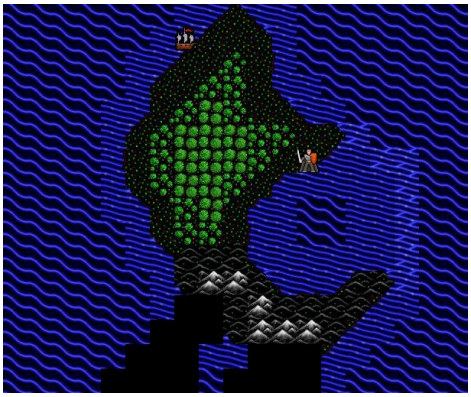
Next, download the patch file from http://bit.ly/1NZkTGz, move the file to the top-level directory of the source code, and apply the patch:

```
$ cd ~/Downloads
$ mv va_list_patch.txt naz-
ghul-0.7.1/
$ cd nazghul-0.7.1/
$ patch -p0 < va_list_patch.txt
```

Finally, build the executable from source and launch the game:

```
$ ./configure
$ make
$ sudo make install
$ haxima.sh
```

To learn more about Haxima Nazghul, visit the home page at http://bit.ly/1FyW7d8. For comments, questions and suggestions, please refer to the original thread at http://bit.ly/1NZklR0.



USING PYTHON WITH THE ODROID-SHOW

SHOWTIME MAKES EVERYTHING EASIER

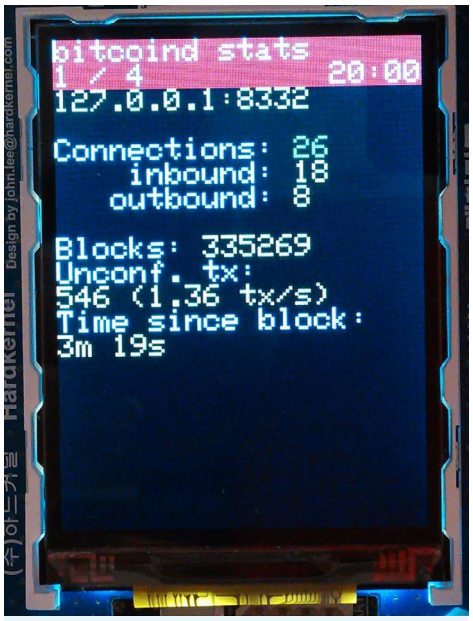
by @Matoking

I created a Python script called SHOWtime for displaying various kinds of information on an ODROID-SHOW using tabs. It may be downloaded from my Github repository at https://github.com/Matoking/SHOWtime. Basically, SHOWtime uses tabs to display information, such as disk and RAM usage, Bitcoin price, disk usage. The shown tab is changed using a set interval, with a default of 15 seconds.

Another interesting detail is that instead of sending the ANSI escape commands manually, I created a convenience class that allows everything to be done neatly using method chaining, like this:

```
ctx.fg_color(Screen.RED).
write("Hello").linebreak().
fg_color(Screen.BLUE).
write("world!")
```

Figures 1 - 2: SHOWTime screenshots



I also included a class named ScreenContext in context.py that allows almost anything to be done on the ODROID-SHOW using Python without having to worry about throttling input or entering escape commands manually. Printing text, changing background/foreground colors, performing linebreaks and most of the functionality can be done easily and neatly using method chaining.

Getting started

Assuming you've created a .py file in the same file as context.py and have performed the steps described in INSTALL, you can start with the following template:

```
from context import Screen, ScreenContext
import atexit

ctx = ScreenContext("/dev/ttyUSB0")

# Make sure the cleanup routine is called to clear the screen
# when we close the script
atexit.register(ctx.cleanup)

# Wait 6 seconds for the screen to boot up before we start uploading anything
ctx.sleep(6).reset_lcd().set_rotation(0)
```

This template creates a new screen context we can use for interacting with the ODROID-SHOW. Note that we sleep for 6 seconds to make sure ODROID-SHOW is done displaying the bootup screen, after which we can be sure that all commands are received and handled correctly. Then, we can start with a simple Hello World program. Insert the following at the end of the script:

```
# Main loop
while True:
    ctx.fg_color(Screen.RED).
write("Hello").linebreak()
    ctx.fg_color(Screen.BLUE).
write("world!").home()
```

This creates a simple loop that displays the text "Hello world!" on the ODROID-SHOW, the word "Hello" in red on the first line, and the word

"world!" in blue on the second line.

The last home() method call makes sure the cursor is placed back at the start, otherwise the words "Hello" and "world!" would be drawn until they were offscreen. Now you can run the script using the Python interpreter. Assuming you named the file example.py, you can just run the following in a Terminal window:

```
$ python example.py
```

Note that you don't need to call sleep() in order to throttle the script's execution to keep the ODROID-SHOW in sync because ScreenContext already takes care of that. However, if you do need it for any reason, you can call ctx.sleep(seconds) to halt the script's execution for any amount of seconds you want. In case you only want to use ScreenContext but not the SHOWtime script itself, you can simply copy context.py, port_open and utils.py and place them in the same directory as your script.

All of the methods in ScreenContext have been commented, so you shouldn't have trouble checking it yourself for what you need. There are, however, some methods which may need some additional demonstration in order to use them as they were intended.

Prevent ghosting

Let's try out the following script.

```
eggs = 555
spam = 1234
while True:
    ctx.write("Eggs %d" % eggs).line-
break()
    ctx.write("Spam %d" % spam).home()
    eggs = 99
    spam = 321
```

Looking at the code, you would expect the screen to display the following text as output:

```
Eggs 99
Spam 321
```

However, since we have to explicitly write over text that has already been displayed to clear it, following is displayed instead:

```
Eggs 995
Spam 3214
```

Fortunately, ScreenContext has a convenient method that prints the given text to the screen and fills the rest of the line with whitespace, effectively preventing these ghosting issues. You can fix the example by doing this:

```
eggs = 555
spam = 1234
while True:
    ctx.write_line("Eggs %d" %
eggs)
    ctx.write_line("Spam %d" %
spam).home()
    eggs = 99
    spam = 321
```

Note that this also removes the need to use linebreak() to change the line.

For more information, or to post questions, comments, or suggestions, please visit the original threads at http://bit.ly/1G7xAa1 and http://bit.ly/1VfzMmW.

PRINCE OF PERSIA

RESCUE THE PRINCESS IN THIS CLASSIC DOS SIDE-SCROLLER

by Tobias Schaaf

Prince of Persia is a much-loved DOS game from the early 1990s. You must avoid deadly traps, solve some simple jumping and environmental puzzles, and engage in sword fights with the guards.

To install Prince of Persia, download the .deb file from http://bit.ly/1LIsPLU, then type the following into a Terminal window:

```
$ sudo apt-get install \
  gdebi xboxdrv
$ cd ~/Downloads
$ sudo gdebi ./sdlpop-odroid*
```

The game may be played with a keyboard or joystick. To use an Xbox 360 joystick, type the following into a new Terminal window before starting Prince of Persia:

```
$ sudo xboxdrv --dpad-only
```

Launch the game by typing the following into a new Terminal window:

```
$ cd /usr/local/share/SDLPoP
$ prince .
```

Press Control-J to enable the joystick, then save the princess!

Prince of Persia was a revolutionary game that used rotoscoped animations

LINUX GAMING: DREAMCAST

SEGA'S LAST BIG CONSOLE COMES TO THE ODROID PLATFORM

by Tobias Schaaf

The SEGA Dreamcast is one of my favorite consoles of all time. And it runs quite well on ODROID devices. Therefore I couldn't help myself but to honor the SEGA Dreamcast by creating an ODROID article about this awesome console, and to give a closer look on what you can expect by playing it on your ODROID device. The SEGA Dreamcast is next to the PlayStation Portable as the most graphically impressive system that ODROID devices can emulate at the moment, with beautiful, stunning graphics and fast-paced action gameplay.

Dreamcast – a little bit of history

The SEGA Dreamcast was the last of the big consoles created by SEGA. Around the 80s and 90s, SEGA and Nintendo were the two major players in the console market and always tried to compete with each other. Nintendo had its Nintendo Entertainment System (NES) which SEGA countered with the Master System. Nintendo had its GameBoy and GameBoy Color, and SEGA had its GameGear. Super Nintendo Entertainment System vs. Genesis/MegaDrive. The battle was tough, and although Nintendo normally had a somewhat a better market, the SEGA systems were often superior when it came to hardware specs. Still in the end, SEGA made a lot of mistakes and developers were annoyed with SEGA's rapid announcement of new consoles (SEGA CD, SEGA 32x, etc.). Therefore, when the SEGA Dreamcast was announced, SEGA actually had trouble finding developers that would support the device, and ended up producing most of the games for the console themselves.

The console was the best you could get in it's time with impressive graphics and even Microsoft Windows CE support. It had a build-in modem and actually was the first console that allowed multiplayer online games, and even further, the first Massive Multiplayer games. Phantasy Star Online was the first game that offered an online community where you could meet other people, form a party, and go on quests together. Even better, it was the first game that came out for different platforms where you could play together with other players.

Soon after the Dreamcast came out, the PlayStation 2 was announced with far superior hardware and worst of all (for SEGA and Nintendo), DVD support. However SEGA and the Dreamcast still had more than a year to establish a market for its games and services before the PlayStation 2 would show up on the market and the rest is pretty much history.

As I mentioned previously, Dreamcast was the last console from SEGA, and afterward they announced they would stop producing any Hardware at all, and instead would create games for other consoles.

In the end, SEGA even produced titles for their big rival Nintendo, and nowadays you can actually play games with Sonic on the Wii or Wii U.

More details

Let's talk more about what you can expect:

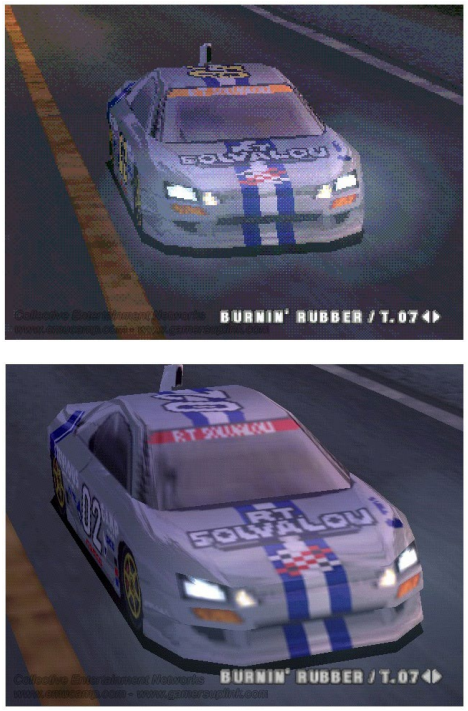
The Dreamcast came out between the Sony PlayStation 1 and PlayStation 2, and that pretty much describes what the Dreamcast is capable of doing. It was far better than a PlayStation 1, but really not as good as a PlayStation 2.

In fact, there was a project to play PlayStation 1 games on the Dreamcast (called Bleemcast) which made the PlayStation 1 games actually look better than on a real PlayStation 1 due to higher resolution and superior graphics powers. This was not a REMAKE of the games, but instead was the game running in an emulator, so the Dreamcast actually had to emulate the games and was still able to improve the graphics.

Games

Besides this, you might want to know what kinds of games you could play on the Dreamcast.

The Dreamcast offered a wide range



Ridge Racer Type 4, running natively on the PlayStation (top), and under the Bleemcast! emulator (bottom)

of games of all genres. In total, over 600 games were released for Dreamcast for different regions (US/Europe/Japan), and some Dreamcast fans are still creating their own games today.

Arcade Games

The Dreamcast is probably most famous for the wide range of Arcade games which were made available for Dreamcast. Prior to the Dreamcast, many consoles aimed to bring arcade games to the people "at home", however Dreamcast actually made this a reality, and in fact turned out to be better than the arcade machines themselves.

Dreamcast brought many games that existed in the Arcades into your home and onto your TV screen, with stunning graphics, great sound, and with up to four players on one console.

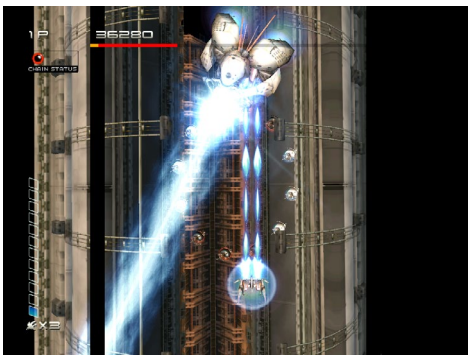
Fighting and Brawler games

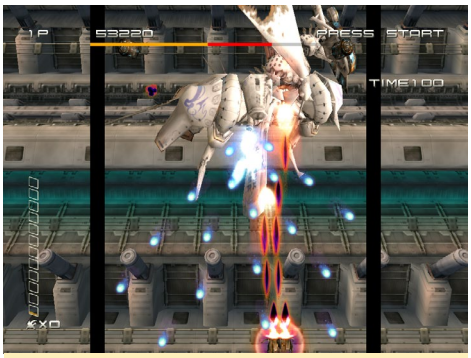
One of the most famous games on Dreamcast is Soul Calibur, which actually turned out to be superior to the arcade version, and on the Dreamcast started a new series of fighting games. While Soul Calibur is now available for many different consoles (including PS3, Xbox 360, and even Android and iOS), most of these games probably wouldn't even exist if it weren't for the success of the Dreamcast version. I personally prefer Soul Calibur over most of the other fighting games, especially over the Tekken series which in my opinion is rather blunt in comparison to the Soul Calibur games. Having swords, staffs, axes, and other weapons to fight each other, and have actually buttons to block an enemy's attack makes the game much more attractive to me then other games of the same genre.

But there are plenty more fighting games on Dreamcast, Power Stone 2 for example, which is a very nice brawler game for up to four players, or the many famous Capcom games like Marvel vs. Capcom 2: New Age of Heroes, King of Fighters, Last Blade 2, Street Fighter games, Dead or Alive 2 (which also had wonderful graphics), Mortal Kombat, and many many more.

But the Dreamcast has more Arcade games to offer than just fighting games and brawlers.

Soul Calibur (top) and Power Stone 2 (bottom) on the ODROID-U3 played through reicast emulator Side Scrollers





Ikaruga

There are many very good arcade side scrollers and shoot 'em ups like Ikaruga or Giga Wing.

Ikaruga is a Japanese arcade shooter for up to two players where you can switch the color of your ship to do extra damage and avoid being hit by the enemy. This shooter has stunning 3D graphics and lighting effects.

GigaWing

Giga Wing is an "old school" arcade shooter which is also for up to two players. There was even a successor: Giga Wing 2, and both play perfectly fine on the ODROID-U3.

There are a lot more games like these, and they are certainly not the only arcade games that were ported to Dreamcast. There is a huge list of Arcade games that were made for Dreamcast including games like House of Dead 2, Virtua Cop 2, Virtua Fighter and so on.

Original Dreamcast games

Arcade games were not the only games SEGA (and other companies) brought to the Dreamcast. There are many titles for the console that were not taken from Arcade machines, or that simply look a lot better on the Dreamcast.

Horror Survival

The Dreamcast had games in every genre, including Horror Survival games like the Resident Evil series. On Dreamcast, this was the famous game "Resident Evil - Code: Veronica", which first came out on Dreamcast, and later on systems like PS2 and GameCube. It was the first of its kind, incorporating real-time 3D backgrounds instead of simply 2D pictures.

But this was not the only game of its kind on Dreamcast. There was also Alone in the Dark: The New Nightmare, Blue Stinger, Carrier, D2, Nightmare Creatures II, and the list goes on and on. If you are a fan of horror survival games, Dreamcast has plenty of games to offer.

Resident Evil: Code Veronica in-game rendering. Claire Redfield facial details (top), first Zombie encounter (bottom)

Role Playing Games (RPGs)

Although not the most common games on Dreamcast, they still had some nice RPG games. Sadly none of the "big players" had their games made for Dreamcast, so you won't find games like Final Fantasy, Tales Saga, or Dragon Quest Saga on Dreamcast. But there are still a few good RPG games for Dreamcast such as the famous Grandia II, Evolution 1 and 2, and Time Stalkers. They might not be the best known RPG games, but nonetheless are really good RPGs mostly found only on Dreamcast (except for Grandia II).

Grandia 2 is a very nice RPG that actually came on two discs, and in order to make the special effects more impressive

Grandia

Evolution

videos were played over the game layer creating a very deep atmosphere and awesome special effects for the "mega attacks". It also offered a very interesting fighting style that allowed you to break an enemies attack before they got to hit you.

Evolution 2 is more of a cute kind of JRPG, with very funny and cute characters. And although RPG games are rare on Dreamcast, the few that are available are quite nice and can keep you busy for many hours. Still with the very impressive CPU and GPU power I wonder what a Final Fantasy game or something similar would have looked like.

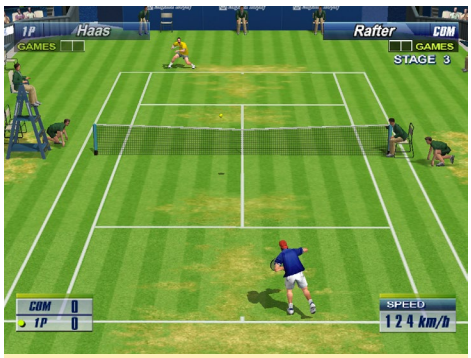
Racing games

I don't want to go too deeply into the details of racing games on Dreamcast. Just enough to say that they definitely exist! There are games out there such as Metropolis Street Racer, Monaco Grand Prix, F335 Challenge, Sega GT, Hydro Thunder, Sega Rally, Test Drive 6, Star Wars Racer, and even the RC car racing game Re-Volt (one of my favorite racing games, not only on Dreamcast). Racing games are fun on the Dreamcast, but the genre is not my personal favorite, so I

won't point out the "best" of them.

Sports Games

Although also not my favorite genre, Dreamcast has quite a number of sports games to offer. Starting with titles from Ready 2 Rumble, NFL, NHL and NBA games, to games like Virtua Tennis 2 and Golf games. There are plenty of sports games available, but the only one that I personally enjoyed playing was Virtua Tennis 2. And although I normally don't like sports games that much, this one was really fun to play, and I played it for many hours on my Dreamcast.



Virtua Tennis 2: One of the few sports games I actually enjoy playing and it's for Dreamcast

Platformers

Every console has this style of game, and platformers (also called Jump 'n Run) is one of the first type of games to come out for every console. The Dreamcast has some really nice platformers such as the famous Rayman 2 and Jet Grind Radio, a game where you play a skating graffiti artist that can do all types

of stunts, a really colorful game.

Honorable mentions

The Dreamcast has many awesome games that I do not want to put into different genres. For example, it actually has a version of Grand Theft Auto 2 (GTA2), and Half-Life was even ported to Dreamcast (together with its expansions Half-Life: Blue Shift and Half-Life: Opposing Force). Several Disney games came out for Dreamcast including Donald Duck Goin' Quackers and Toy Story 2. Quake 3 Arena, Unreal Tournament, Railroad Tycoon 2, Worms Armageddon, and Worms World Party can even be found on Dreamcast. The list of games is very impressive considering the fact that the console was only was on the market from late 1998 until 2002, at which time SEGA announced that they would stop producing new devices.

Games that defined Dreamcast

There were a few games on Dreamcast that really defined gaming history and were outstanding for the Dreamcast. One of these games I already mentioned: Soul Calibur, and this series for consoles was born on Dreamcast and really was a showcase for the Dreamcast console. The graphics were really impressive, butter smooth, and showed just what this console was capable off. It was a port from the Arcade machine to a home console and even exceeded the original Arcade machine version. It still counts as one of the best games in gaming history.

Another game that was really made for the Dreamcast is Sonic Adventure and it's successor Sonic Adventure 2. There were some attempts to bring Sonic to the 3D world, but only Dreamcast was really able to produce a game that had everything that you would expect from a Sonic game: Sonic, Rings, awesome music, and most important of all SPEED! The game gives you really that feeling of speed, especially when you get spun around in looping or flying through the air. You find everything you are used to from the 2D game back here in 3D: the checkpoints, the extra boxes with rings, shields, faster running, and so on. The second game even allowed you to play as the bad guys (how awe-

Rayman 2 and Jet Grind Radio two very fun to play Platformer on the Dreamcast

GTA 2 on the Dreamcast: Movement is somewhat difficult however

Sonic Adventure 2: Surfing around the corner catching rings, or running fast enough to run along a wall are just two of the things you can do.

some is that?!).

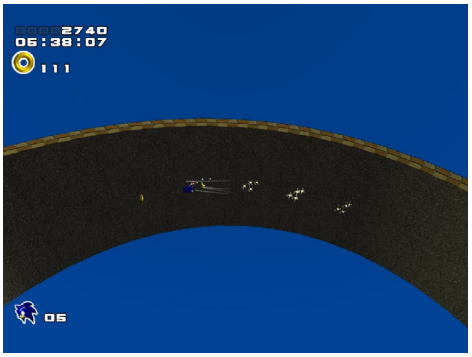
Another game that is known as one of the best games on Dreamcast is Crazy Taxi and Crazy Taxi 2. Both came out as Arcade games first as well, but the ports to Dreamcast are really good and also had some extra content. The soundtrack was simply amazing, and the game really fast and fun to play. Where else do you get to drive with your car through the park, or jump over cars and your fare actually like that crazy sh*t?

But fast action and racing were not the only great Dreamcast games. The few RPGs that actually were on Dreamcast are quite impressive as well. Skies of Arcadia is one very impressive RPG where you play as Vyse, an "Air Pirate" in a universe inspired by Jules Verne. You travel between flying islands, and fight against monsters and the soldiers of the Valuan Empire. This game has it all: good characters with a nice background story, character progression, upgradeable weapons and ships, air to air combat between ships, impressive spells, a very deep story, and oh did I mention AIR PIRATES?!

But this is not where it stops when it comes to the Dreamcast. I mentioned earlier that Dreamcast had an integrated modem that allowed you to play some games online with friends. And this also opened up the world of consoles for the first MMO games. Phantasy Star Online was the first cross-platform multiplayer game where you could meet up in a lobby with your game character, and then venture into the wild to solve quests together with friends. It even offered a bank for you to store your money and items. For it's time, this game was very good and had decent graphics, and some people still play it today on private servers. It's a very nice RPG where you don't have to concentrate on how to set your character points (like strength and agility), but rather can concentrate on leveling your skills by using them in combat. Level up your weapons and your abilities to use them. The job you choose defines how your character will progress. This game was a major success on the Dreamcast, and opened up the market for MMOs on consoles and other platforms. It had several successors like Phantasy Star Online v2 (also on Dreamcast), or Phantasy Star Blue Shift (on Windows), and some years later the Phantasy Star Universe series for PC, PS2 and Xbox360.

The last title (or better yet, series)

Sonic Adventure 2: Sliding down a railing and running through looping things that really make your feel the speed.

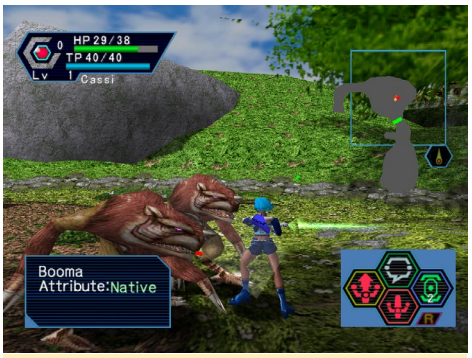


Crazy Taxi: Grab a fare and bring it to the desired destination in any way possible

Skies of Arcadia: Very nice graphics, round based fights with a complex skill and energy system, as well as ship to ship combat make this a really awesome RPG game on the Dreamcast.

Skies of Arcadia: Really nice effects for all of these special attacks, right from the start of the game.

Phantasy Star Online: Character creation screen (top) first visit on Pioneer I space station (bottom). PSO has a lot of interesting things to offer, some of which made gaming history.



Phantasy Star Online: Weapon and item shop on the space station (top) fighting monsters on a mission (bottom). PSO had really good graphics for an MMO of the year 1999.

that is really worth mentioning is the Shenmue series. Although not 100% an "open world" game, this is as close as you can get with something like this back in the year 1999. Shenmue was an very impressive title. It's was basically p*rn for all the game freaks out there that loved really deep RPG and Adventure games. This game was so deep that it's hard to grasp all the things that you can do in the game.

You are Ryo Hazuki, a martial art student on a quest for revenge of your father's murder. This game is extremely deep. The Story is huge and was actually designed to be a trilogy. The game offers day and night cycles as well as different weather. You have to manage your life: go working to earn money, attend to your social contacts, you can gamble, have tons of mini games, or simply go shopping. Eating, drinking, and talking to people to ask them for directions. There is almost nothing you can't do in this game. Even tasks like feeding a little kitten have to be done in this game. While the game progresses, you have to fight against other people, learn new combat moves from strangers and friends, and have to perform quick events when you fight or follow some people. This game is one of the most expensive games in the history of game production (an estimated 47~70 million USD), and you can see that the money was well used in this game. The places feel alive: you always have many people around you doing their daily business, you can talk to them, ask for directions, and interact with them in many ways. This series was praised by critics, and can be found on many of the "greatest video games of all time" lists. The series was actually stopped after Shenmue 2, leaving the ending open. , but was recently announced to finally get a finish with the Kickstarter financed successor Shenmue 3 from the maker of Shenmue 1 and 2. It's suppose to be released for PS4 and PC in December 2017.

However it was recently announced that the series would finally be completed with the Kickstarter financed successor 'Shenmue 3' from the maker of Shenmue 1 and 2. It is expected to be released for the PS4 and PC in December 2017.

Shenmue 2: Facial details are very impressive, and some hair is even individual rendered (top). A lively plaza is seen here (bottom), it is not rare to see places like this, with birds and many people in the same place. Dreamcast could manage all of this.

Shenmue I: Lan Di (the murderer of Ryo's father) holding Ryo up in the air in the the opening of the game (top). Keeping your money together is not always easy (bottom).

ODROID Magazine is now on Reddit!

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

Dreamcast on ODROID

Lots and lots of people are talking about SEGA Dreamcast, but you might wonder: how well does it work on ODROID devices?

All the pictures in this article (except the ones about PS1 and Bleemcast) are directly taken from a ODROID-U3 running reicast as an emulator. Lots of games are running fine, some have some issues, and others won't run at all. But the ratio is still rather good. If I would have to take a guess, I would say that 60% to 75% of all SEGA Dreamcast games are working on reicast, and therefore on ODROID devices.

The most common issues I encountered where graphics glitches. It seems that fog and LOD is not always working correctly. Far away objects showing strange patterns instead of a fog slowly letting them disappear. Some games have a few sound issues. And nearly all games have issues playing the videos at full speed, which is actually rather strange since the videos are very low resolution.

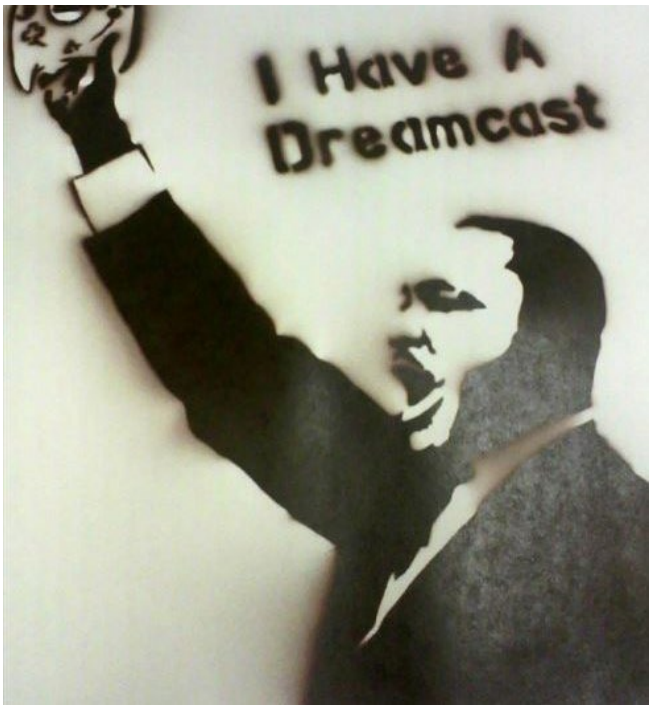
Holzhaus from the ODROID community takes a major part in developing reicast, and actually was able to integrate ODROID support in the upstream version of reicast.

Together with some other fixes, the current version of reicast runs really well on our ODROIDs (I only had to make minor changes to get it to work the way I wanted it).

I'm looking forward to further improvements of the emulator and will update the packages in my repository as often as possible to keep all "the good stuff" coming to our ODROIDs.

Next to the PSP emulator, this has become one of my favorite emulators for the ODROIDs that allows me to replay my favorite Dreamcast games without my having to get the old console out of the basement and try to figure out how to attach it the my current TV. It also allows me to play these awesome games with my friends, which makes it even better than the PSP, because now I can fight in Soul Calibur or Power Stone against my friends and have a friendly competition match with them.

Dreamcast games are a nice addition to the already impressive number of emulators and games that run on ODROID devices and I can only suggest to everyone that likes playing games that they check out these awesome Dreamcast games on the ODROID platform.



MEET AN ODROIDIAN

WILLIAM HENNING (@MIKRONAUTS) ROBOTICS AUTHORITY AND PROLIFIC TECH BLOGGER

edited by Rob Roy

Our man William inspires us to do more with our ODROIDs!

Please tell us a little about yourself.

My name is Bill, and I am addicted to robotics & electronics. I live in Langley, a suburb of Vancouver, in beautiful British Columbia, Canada, with my lovely wife Agnes, otherwise known as "Wifey". I am the owner, CEO, CTO and chief bottle washer of Mikronauts (http://www.mikronauts.com), which is a consultancy specializing in custom software and hardware solutions largely for industrial control clients, which is not very surprising, as I have over 30 years experience in industrial control as a programmer/analyst, systems analyst, systems architect and technical project manager. I also design and sell robotics and educational products. We have a whole passel of "stinky boys" (nephews, named as such by wifey) and one "non-stinky" niece in the family, whom we love to spend time with.

How did you get started with computers?

I started with an HP mini-computer/calculator with a punched card reader, tape drive and line printer, running an HP basic in grade 10. Later, the school bought some original Apple][computers for the lab. I wire-wrapped a Z80 board, and used a Cosmac-Elf, Kim-1 and other single board computers of the time. We had a great physics/electronics teacher!

After getting some part-time jobs, I saved my pennies and bought an Atari 400 at first, then an Apple][clone with a Z80 card. I went to Simon Fraser University for Computer Science, where I also worked for the Department of Education, and bought one of the first Amiga 1000 computers in Vancouver. I was blown away by the graphics demos, and the graphical desktop with multi-tasking in 512KB with a floppy drive. Shortly after that, I wrote a caching driver for the "Wedge", which was a pc/xt RLL controller interfaced to the Amiga by another Amiga user. I also ported a Valgol compiler to the MC68000, and generally had a ton of fun with it.

Later, I wrote the software to drive the laser disc players and video genlock devices for the "Amiga Theatre" at Expo 86 in Vancouver. It was a blast to see a whole wall of TVs running synchronized video shows from laser discs controlled by an Amiga.

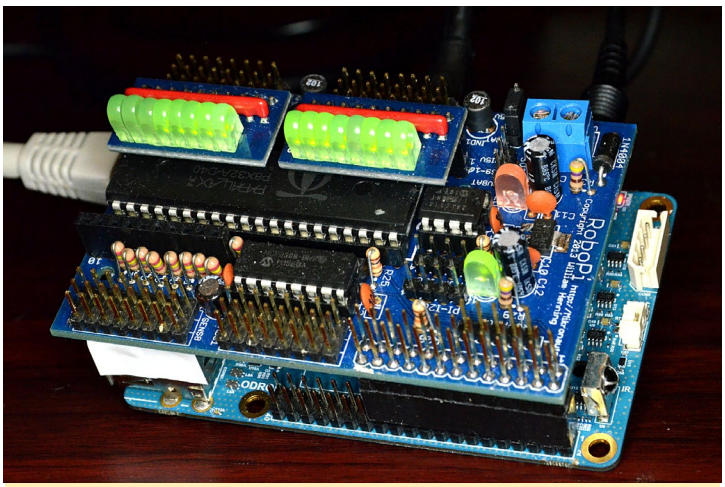
In 1984, I started working for Pan-Abode, a log home manufacturer in Richmond, BC, where I started my career in industrial control. I wrote software for controlling the drying of logs in large kilns, designed voice controlled "Smart House" software, and wrote Lisp extensions for AutoCad, including solar energy modelling of the heat gain and loss of log homes depending on window sizes, orientation, and home design plans.

Later, I worked for Universal Dynamics, designing software and hardware solutions for large industrial clients in North America, such as power utilities, mills, grain terminals, smelters and more. In the late 1990s, I became a consultant and also started two web sites, that later became extremely popular:

CPUReview.com, where I reviewed PC processors, motherboards, video cards and experimented with overclocking. I used to write launch reviews for new Intel and AMD processors by getting them under NDA from Intel and AMD early in order to have the reviews be ready in time for the product launch.

AboutLinux.com, where I reviewed new Linux distributions, Linux software, and wrote how-to guides.

After the .Bomb collapse of internet advertising, I stopped updating both of the sites, and unfortunately I lost both domain names in 2001. I never received renewal notices, and since I ran my own web and DNS servers, I did not notice that domain squatters had snapped them up. I then taught an advanced networking lab at the British Columbia Institute



RoboPi being tested on an ODROID-C1 for a C1 Review

of Technology, where I later did research on industrial control network security, and consulted for a startup on a network security appliance.

In 2006, I read about a new microcontroller called the Parallax Propeller that had a very interesting architecture, and was way ahead of its time. With eight 32 bit RISC cores, local memories and a 32KB shared memory, it was far more power than the original mini computers, in a handy 40 pin dip package. I started Mikronauts.com to blog about the Propeller and my electronics experiments. I had a vision of running a stripped-down, small version of Unix on the Propeller, but the architecture limited programs to at most 506 instructions in length. To get past this limitation, I came up with a "Large Memory Model" for the Propeller (LMM was a tongue-in-cheek nod to the large memory model of 8086 compilers) that used a self-modifying fetch-execute loop containing as little as four instructions in a cog in order to implement a virtual machine to allow executing programs from the shared memory. 32KB allowed for 8192 instructions, a factor of 16 increase in size over what could run in a cog! I started writing an operating system for LMM code, and an LMM assembler, but unfortunately I did not have time to port a C compiler to fulfill my idea of running a small Unix on the Propeller.

I did, however, design several single board computers around the propeller, including Morpheus, which was a dual Propeller machine. One Propeller was used primarily for I/O, and the second was used for high resolution bitmapped graphics. It had 512KB of external SRAM, expandable to 16MB, and swap space. Another project was called PropCade, which was used for VT100 emulation and retro-gaming.

Later, I consulted for Parallax for their GCC port to the Propeller, which uses LMM. When the Raspberry Pi came out, I gave up on designing full-fledged computers based on the Parallax Propeller, as there was no way to come even close to the price/performance ratio of the Raspberry Pi. I knew that the Propeller would make an excellent hard real time I/O expander for the Raspberry Pi, so I designed the Propeller based RoboPi advanced robot controller board for the Raspberry Pi.

As I started to really get involved with the low-cost ARM based single board computers that were appearing in ever greater numbers on the market, I decided to re-position Mikronauts as a site that reviews single board computers, and began to publish articles on robotics and electronics projects featuring single board computers, robots, and Mikronauts products.

What attracted you to the ODROID platform?

I found a thread about the ODROID-W on the Raspberry Pi forums. The ODROID-W looked like it would make a great embedded module for robotics and industrial control. Due to its Raspberry Pi compatibility, it seemed like a great match for RoboPi. Unfortunately, shortly after I received my order of ODROID-W goodies (three W's, two LCD's, other expansion modules), it was discontinued, well before I could write a review of it.

Fortunately, Hardkernel announced the ODROID-C1 shortly thereafter, and I immediately ordered six of them along with a bunch of accessories. This was well before the Raspberry Pi Model 2 came out, and how could you go wrong with a quad core ARM SBC for $35? This time, I finished the review! I found that the ODROID-C1 greatly outperformed the original Raspberry Pi's.

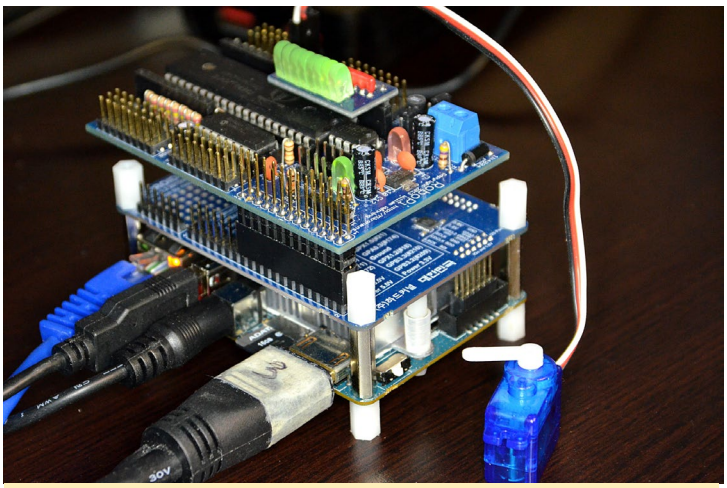
Months later, after the Raspberry Pi Foundation released the Raspberry Pi 2 Model B, the ODROID-C1 still kept its performance advantage. When Hardkernel released the XU3, I was very tempted to get one. Unfortunately, the price was too high to be considered as a low cost ARM board, when compared to the ODROID-C1, Raspberry Pi 2, Banana Pi and many other ARM boards. I loved the feature set, but the less costly SBC's met my needs.

Recently, when Ameridroid asked if I'd be interested in reviewing the new ODROID-XU4, I was definitely interested. Based on what I had been reading about the eight core big.LITTLE ARM chips, and with a price less than half of the XU3, I thought the XU4 might have had the price/performance ratio needed to justify its higher price, and I was right. The performance is outstanding.

How do you use your ODROIDs?

Right now, I use my C1s as small desktop replacements and media players. I keep meaning to make a C1 + RoboPi based robot, however, I am concerned with the current consumption when the C1 is powered off. I recently received a suggestion for how to control that from one of the administrators on the ODROID forums, but have not tested it yet.

(Figure 4 - My CAD and software development workstation, shown while working on HexPi)



RoboPi being tested on the ODROD-XU4 for an XU4 review

I have switched my RoboPi C library development to the XU4, since it provides a much faster compile test cycle then a Raspberry Pi, and the libraries and executables compiled on it work on my other ARM v7-based boards. I have a few other uses in mind for my C1s, which you will see over the coming months at www.mikronauts.com, and perhaps here in ODROID magazine.

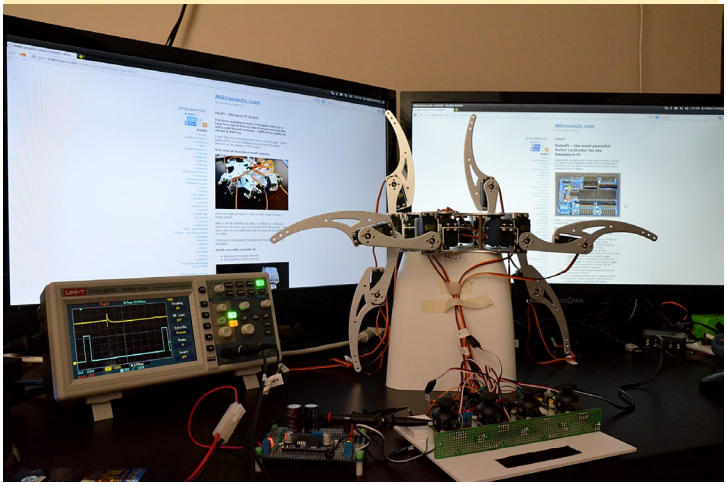
Which ODROID is your favorite?

Currently the XU4 is my favourite ODROID. It is the fastest ARM board I have for compiling and testing ARM v7 code, and there's no need to bother cross-compiling from a PC. It also makes a great desktop replacement, and it does extremely well with Kodi as well. The C1 is a close second, and perhaps a better choice for applications that don't need as much speed, and need more miserly power consumption.

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

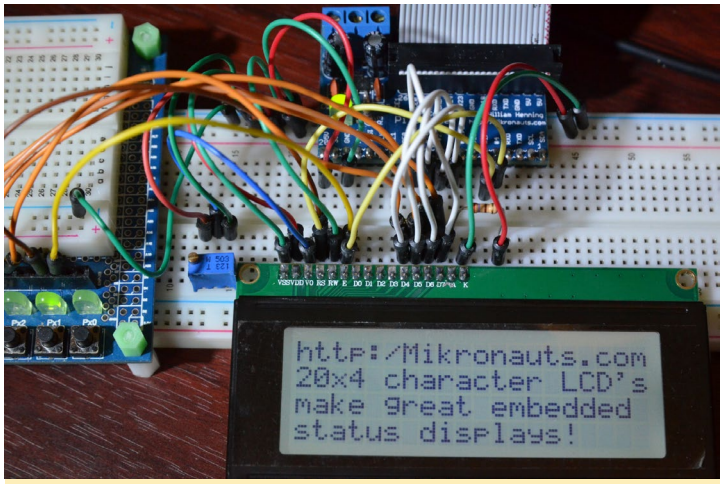
Yes. I work with many micro-controllers, SBCs and computers. You can expect many more IoT, Robotics and electronics projects from Mikronauts, including projects for ODROID boards. I am developing more products aimed at the robotics and educational markets, and will be testing them for compatibility with the ODROID-C1 and ODROID-XU4.

William's CAD and software development workstation, shown while working on HexPi

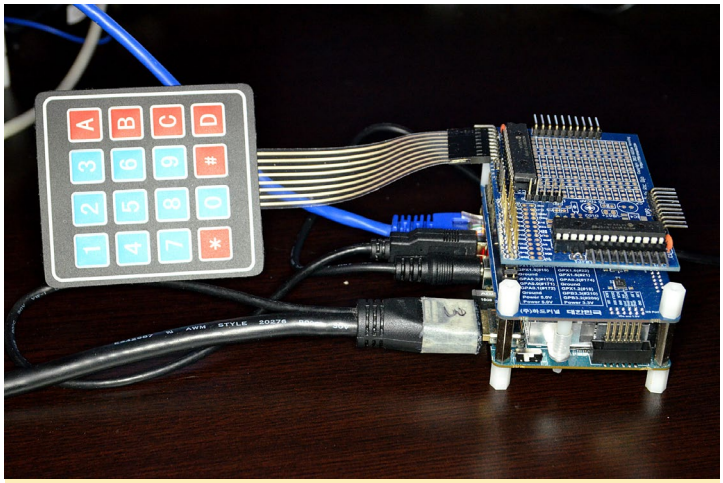


I have a number of projects in the Raspberry Pi section of my site that should work on the ODROID-C1 and ODROID-XU4 with Shifter shield simply by changing GPIO numbers. If there is enough interest, I would be happy to publish C1 and XU4 adaptations of the following:

20x04 and I6x02 LCD interfacing (with Python library)
4x4 matrix keyboard interfacing with I2C I/O (with Python library)
24 channel I2 bit data acquisition board (with Python library) expandable to 64 channels



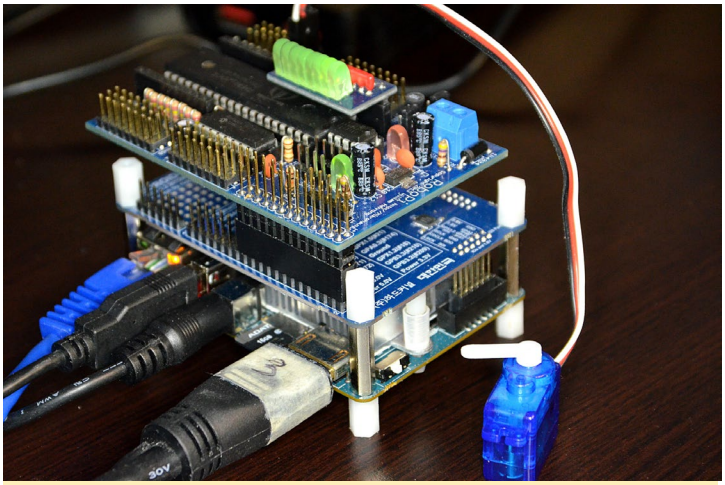
Interfacing 20x4 and I6x2 LCD's, complete with Python library



4x4 matrix keyboad interface project using I2C I/O expanders

Some Mikronauts products are not fully compatible with the C1 and XU4 due to the differences in a few pins of the 40 pin GPIO connector, such as the 1.8V analog inputs. Some also depend on software that is not available on the C1 and

XU4, like the pigpio library and servoblaster kernel driver. Don't get me wrong, because I like having the analog inputs on the ODROID boards, and intend to use them in some projects "Real Soon Now"!



24 channel Data Acquisition project using 3 GPIO, de-multiplexer and I SPI chip select

What hobbies and interests do you have aside from computers?

Wifey – I mention her because I am not an idiot!
Travelling – my favorite trip is flying to Hawaii, staying on Waikiki beach for a week, and cruising around the islands
Photography – I even went professional for a while earlier this millennium
Reading - mostly science fiction, because I do a lot of reading on my trips
Movies and TV - I am a big fan of science fiction, action/adventure and comedies
Family – catching up, playing with and teaching the rug rats in the family about robots and computers
Food – eating the goodies that wifey makes for me!

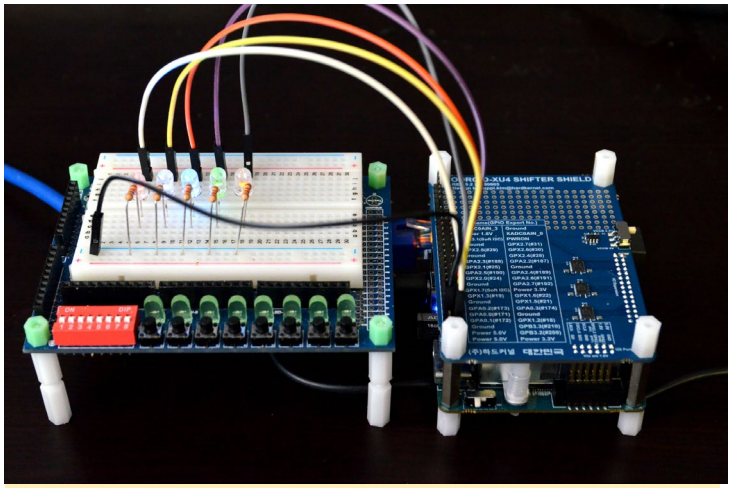
What type of hardware innovations would you like to see for future Hardkernel boards?

I'd like to see USB3.0, SATA, DPI, MPI, better Gig-E, more memory and GPIO, and dual-head support. I don't ask for much.

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

Find a project you want to do, then pick a language, dive in and do it! Remember that Google is your friend, and use it to find other similar projects to what you want to do, and follow their examples on how to actually make your project work. There are a lot of excellent examples out there, and some not so great ones too.

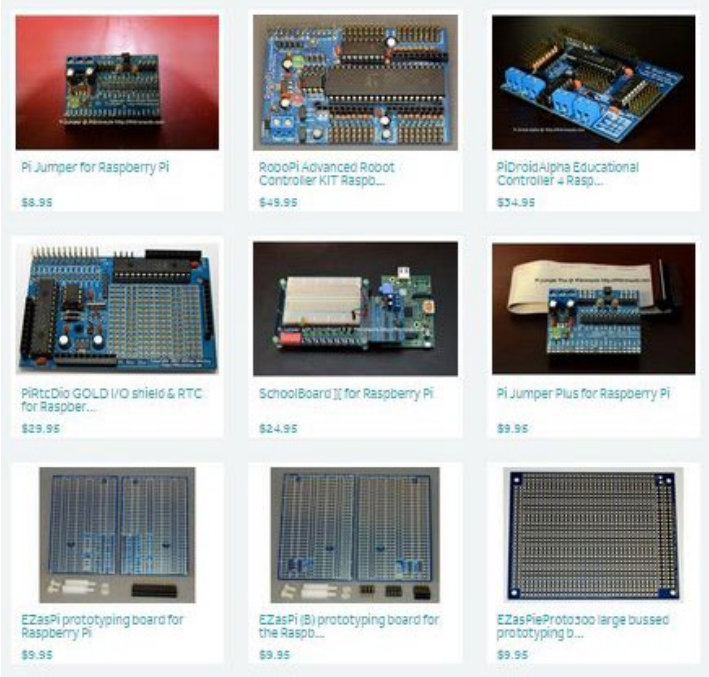
For beginners, I'd recommend Python to start with. I don't think flowchart and graphical tools are very useful for teaching, except at the elementary level. If you get hooked, and want to learn more, I think C is still easier to learn than C++ and its alternatives. Of course, if you really want to understand computers, at some point you should learn assembly language and electronics. Learn to solder, then build your own boards from scratch.



SchoolBoard][development board with ODROID-XU4 Shifter Shield and five superbright LED's

Don't believe anyone who tells you that you do not need to understand low-level programming. If you don't understand assembly language and pointers, no amount of C++ templates and libraries will help you understand why your embedded code is not working right. You will need oscilloscopes, logic analyzers and a good understanding of how it all works to really be able to debug embedded code. And make sure to have fun!

Check out Williams' Mikronauts eBay or Tindy Stores by visiting www.mikronauts.com



ARM TechCon

November 10-12, 2015
Santa Clara Convention Center | Santa Clara, CA

HOME | REGISTRATION | EXPO | CONFERENCE | ABOUT | RESOURCE CENTER

Conference Overview

Get Ready for ARM TechCon 2015, November 10-12 in Santa Clara, CA!

ARM TechCon Conference Hours

Tue, November 10: 8:30 am – 4:20 pm
Wed, November 11: 8:30 am – 5:30 pm
Thu, November 12: 8:30 am – 5:30 pm

ARM TechCon Expo Hours

Tue, November 10: Expo Floor Closed
Wed, November 11: 11:00 am – 6:30 pm
**Happy hour: 5:30 pm to 6:30 pm*
Thu, November 12: 11:00 am – 6:30 pm
**Happy hour: 5:30 pm to 6:30 pm*

HARDKERNEL

2015 Event At A Glance

ARM TechCon 2015 Event At-A-Glance

Why Attend? | Who Attends ARM Techcon? | Tracks & Topics

Why Attend?

Join us at ARM TechCon 2015 and:

- Get exclusive access to trending design strategies, methodologies, and tools for building ARM®-based products through technical sessions, hands-on labs, exhibits, demonstrations and keynote and panel discussions.
- Network with fellow design engineers and software developers shaping the future with ARM technology.
- Make a major impact on tomorrow's technology when you collaborate directly with ARM® Ecosystem Partners that are part of an exclusive deep and diverse ARM Connected Community®.
- Meet with ARM Executives, System Architects, and Fellows to assist you with your ARM-based strategies.

REGISTRATION

CONNECT WITH ARM TECHCON

Hashtag: #ARMTechCon

2015 SPONSORS

Diamond Sponsors

cadence | IAR SYSTEMS

Platinum Sponsors

freescale | Mentor Graphics | SYNOPSYS Silicon to Software

Silver Sponsors

applied micro | Atmel | Rambus | life.augmented