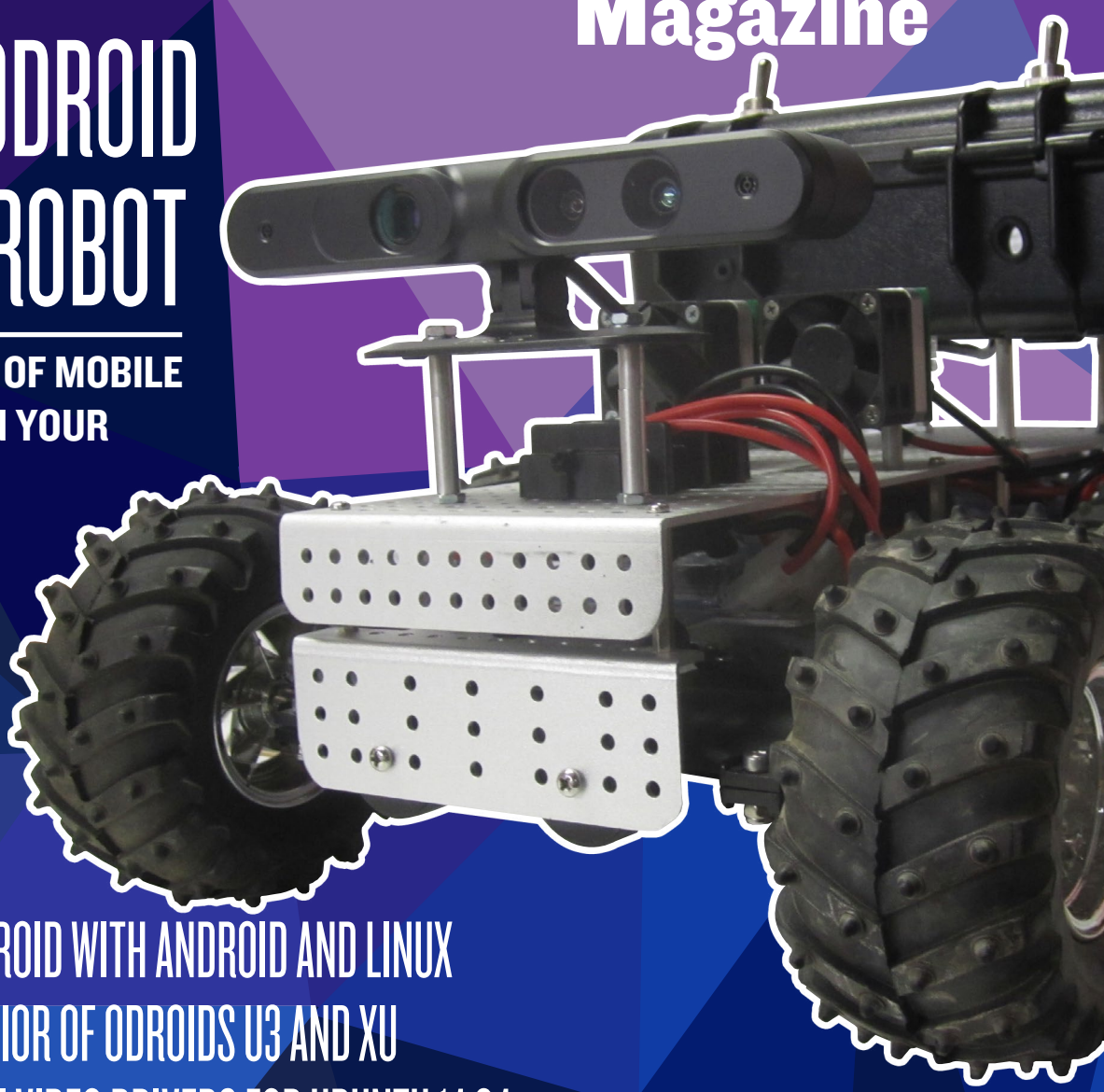
Android Booting Process • Compile and play the classic game DOOM

ODROID Magazine

Year One
Issue #5
May 2014

BUILD AN ODROID POWERED ROBOT

THE REAL MEANING OF MOBILE COMPUTING WITHIN YOUR REACH AT LAST!



AND ALSO:

- DUAL BOOT YOUR ODROID WITH ANDROID AND LINUX
- THE THERMAL BEHAVIOR OF ODROIDS U3 AND XU
- RECOMPILE THE MALI VIDEO DRIVERS FOR UBUNTU 14.04
- OS SPOTLIGHT: FULLY LOADED

HARDKERNEL DOES IT AGAIN! EXPAND YOUR ODROID U3 WITH:

ODROID-UPS: UNINTERRUPTABLE POWER SUPPLY

ODROID-SHOW: AN ARDUINO COMPATIBLE DEVICE

What we stand for.

We strive to symbolize the edge 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 devices to EU countries! Come and visit our online store to shop!

Address: Max-Pollin-Straße 1
85104 Pförring Germany

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

Our ODROID products can be found at:
`http://www.pollin.de/shop/suchergebnis.html?S_TEXT=odroid&log=internal`

Our DIY edition last month was one of the most popular issues we've had so far, and really showed off how flexible and useful the ODROID boards can be in realizing affordable high-end Maker projects in your home. The latest exciting news comes from Chris McMurrough, who publishes the Ubuntu Robotics Edition on the ODROID forms, and is very talented when it comes to robotics and hardware automation. His Off-road Unmanned Ground Vehicle Robot, which looks like it comes straight from NASA's space program, demonstrates how easy it can be to build your own robot using an ODROID and a few readily available parts.

Hardkernel has once again created a couple very useful peripherals for the ODROID-U3: the ODROID-SHOW and ODROID-UPS. The ODROID-SHOW is a 2.2" 320x240 pixel LCD panel that connects to any computer, and is capable of displaying text, statistics, images, and other useful real-time information. It fits neatly on top of the ODROID-U3, is reasonably priced, and provides a more robust alternative to the Arduino One's 2-line text display.

The ODROID-UPS (Uninterruptible Power Supply) is designed to keep your mission-critical applications running during power failure. Instead of using an expensive and bulky battery backup, the ODROID-UPS fits in the palm of your hand, and can keep the computer running for up to 4 hours without recharging, depending on the processing load. Our article also includes a simple script example for shutting down the ODROID safely when AC power is no longer available. Both the ODROID-SHOW and ODROID-UPS are available from the Hardkernel store at `http://hardkernel.com/main/shop/good_list.php`.

Also featured in this issue is a beginner's guide to flashing prebuilt OS image files, which can be challenging for those who are used to the typical PC "installation disk" method. Image files provide a convenient way to install an entire operating system in a single step, and save hours of configuration by getting your ODROID up and running quickly. For more advanced users, we also present a highly requested feature: How to setup an Android/Ubuntu dual boot system on a single hard drive.

Our regular columnists have been hard at work creating step-by-step guides for software enthusiasts. Tobias presents a comprehensive guide on compiling your favorite games such as Doom, I outline the features of the popular Fully Loaded community image, Jussi shows us exactly how the ODROID-U3 and ODROID-XU compare under heavy load, and Nanik helps us understand how the Android boot process works. We will continue to color-code the article titles in order to categorize them as Beginner, Intermediate and Expert material, to make sure that there is always something for everyone!

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
Makers of the ODROID family of quad-core development boards and the world's first ARM big.LITTLE architecture based single board computer.
Join the ODROID community with members from over 135 countries, at http://forum.odroid.com, and explore the new technologies offered by Hardkernel at http://www.hardkernel.com.

HARDKERNEL

ODROID Magazine

Rob Roy, Chief Editor

I am a computer programmer living and working in San Francisco, CA, designing and building websites. 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 cluster of 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 and images at `http://bit.ly/1fsaXQs`.

Bo Lechnowsky, Editor

I am President of Respectech, Inc., a technology consultancy in Ukiah, CA, USA that I founded in 2001. From my background in electronics and computer programming, I manage a team of technologists, plus develop custom solutions for companies ranging from small businesses to worldwide corporations. ODROIDs are one of the weapons in my arsenal for tackling these projects. My favorite development languages are Rebol and Red, both of which run fabulously on ARM-based systems like the ODROID-U3. Regarding hobbies, if you need some, I'd be happy to give you some of mine as I have too many. That would help me to have more time to spend with my wonderful wife of 23 years and my four beautiful children.

Bruno Doiche, Art Editor

Is now contemplating to go into a coffee abstinence rehab after discovering H20 airpress coffee and drinking it by buckets. Also managing a black belt on multinational reorganizations. Just don't ask him what his job is right now as he can't explain it even for himself. Only that it involves lots and lots of SAN storage devices and tons of bureaucracy.

News from Art Editor Bruno:

And the changes keep going! The forums ask, we attend when it makes sense to make it done. We got a bunch of requests to eliminate the green screen terminals to make it easy to print the magazine. Now we unified the monospaced code on the yellow blocks. Together with our new shift column scheme, it came quite well. Also, to make even more clear which level each article is about, we will be placing a star to differentiate the skill required for each article.

There is also elasticity on the monospace font on some code blocks, where we are now using a smaller type to get things a little neater (albeit harder to edit) and easier to read. believe me, I'm the guy that gets the most frustration on the little graphic issues.

As proof that there are all kinds of synchronicity on the forums, I saw a tip from a user that uses fortune |cowsay on his bashrc file to greet the user. That was a tip that was reserved for the Linux tips column this month, but didn't make it into this edition. I'll leave you with a motto that was a mantra for me this month.

```
 _____________________________________
/ You'd like to do it instantaneously, \
\ but that's too slow.                 /
 -------------------------------------
  \
   \____
  /@    ~-.
  \/ __ .- |
   // //  @
```

INDEX

ANDROID BOOTING PROCESS

UNDERSTAND THE INNARDS OF HOW YOUR ODROID BOOTS UP ANDROID

by Nanik Tolaram

All computer platforms, including the ODROID-U3, have some predefined way of booting up in order to load the operating system. For example, in a conventional PC, the BIOS will be the first binary that gets executed, and the subsequent chain of boot sequence events are controlled by the microprocessor when power is applied to the device. Different processor architectures have their own ways of booting, and ARM processors power up in a different way than an x86 processor. In this article, we will look at what happens when you plug in the ODROID-U3 device up until the time that the Android screen appears.

Figure 1 illustrates, at a high level, what is happening during the boot process. We are going to use this diagram to go into detail on each of the steps.

Boot ROM

This is the first program that is run by the microprocessor when it starts up, and resides "inside" the processor, usually installed by the manufacturer. Normally, this program runs in the tens of kilobytes and it's main function is to setup the hardware to a particular state that will enable the next step of the boot process to continue. This ROM is a bit of a "black box" for developers, as the source code is generally not available due to proprietary restrictions. The ROM is responsible for making sure that booting devices are initialized and ready to be used, as the next stage will begin by reading from the external storage (SD Card, eMMC, etc) in order to continue the boot process.

Bootloader

Initializing the hardware at the Boot ROM stage is crucial for the bootloader, since the bootloader resides inside a storage device. In the case of the ODROID-U3, this device is the SD card or eMMC module. During the execution of the ROM code, it will read the bootloader binaries from the storage device and start executing the loaded code.

As can be seen in Figure 2, for the ODROID-U3 platform, there are couple of files that are used as bootloaders. The bl1.bin files are proprietary bootloaders from Samsung containing a number of activities that initialize the hardware even further. The u-boot bootloader source is available for ODROID-U3 and can be easily modified and recompiled. The 2 main binaries (bl1.bin and bl2.bin) are files that have been signed and are required for booting. The file bl1.bin will be the first to be looked up and executed when the Boot ROM completes its execution.

On completion of bl1.bin, the code will look up bl2.bin and continue from that point on. We are not going to discuss what exactly bl1.bin is doing since it is unknown to anyone outside Samsung. The other file, bl2.bin, is generated as part of building u-boot. However, this file needs to be signed by Hardkernel in order to work with the ODROID-U3.

Figure 1 : High level boot process

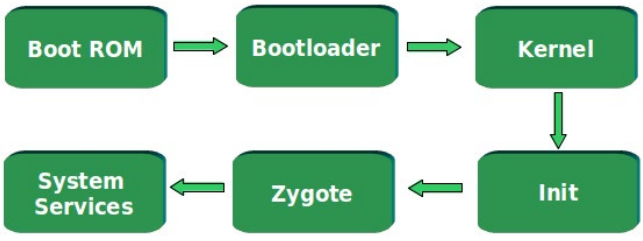


Figure 2 : SD card bootloader layout

Finally, the tzsw.bin file is a proprietary file from Samsung/ARM that allows code to be run in a secured zone.

Kernel

Once u-boot completes successfully, it will load the Linux kernel and run it. The executed kernel is the same kernel that is used to boot up Ubuntu or any other Linux distro, but contains Android-specific drivers that are needed to run the Android stack. At this stage, there is nothing special going on, as it is the same kernel that you normally see when booting Linux.

Init

When the Linux kernel has completed all of the initialization routines, and

Figure 3 - The Init application

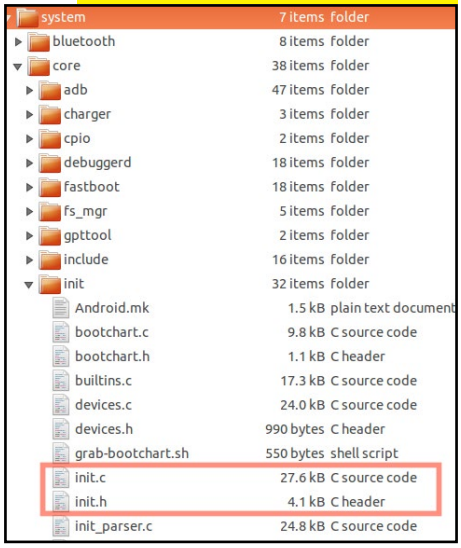


Figure 4 - Other .rc files inside init.rc

```
import /init.${ro.hardware}.rc
import /init.usb.rc
import /init.trace.rc
```

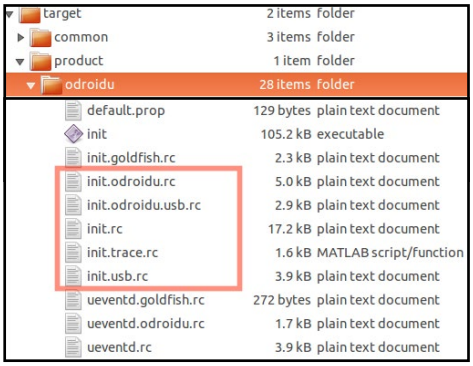


Figure 5 - Complete list of init .rc files for the ODROID-U3

all the services are running, it will run a program called init. From this point forward everything that runs is related to Android, or what is known in the Linux world as the userspace layer, from loading the HAL (Hardware Abstraction Layer) drivers all the way to running your application. The init application in Android is different compared to a normal Linux environment, and resides inside the /system folder of the Android source code.

The primary function of the init application is to initialize all the necessary directories, permissions, services and environment variables and properties. The init application operates by reading a configuration file called init.rc inside the out/target/product/odroidu/root directory. The init.rc simply includes other .rc files.

The init.rc contains what is known as Android Init Language. Explanations of the different available commands can be found at `http://bit.ly/1kfCibb`. Let's explore the contents of the .rc files in detail.

```
on boot
    setprop ro.build.product odroidu
    setprop ro.product.device odroidu
    setprop ro.radio.noril yes
    setprop ro.kernel.android.ril 0
    setprop ril.work 0

# Run sysinit
    start sysinit
```

The above snippet is taken from the init.odroid.rc file under the on boot command. Its purpose is to set different environment properties for ro.build.product, ro.radio.noril, and other services. These different environment variables are used internally by the Android framework as part of the initialization process. A service called sysinit is also started. The following list shows the execution command sequence when init completes reading the init.rc:

```
early-init
init
early-fs
```

PIPE VIEWER
GET MORE INTERACTIVE WITH YOUR DATA PROGRESS TOOLS

by Bruno Doiche

Tired of getting no output on dd whenever you need to backup or flash your eMMC?
Then install pv, it is a program that you can put between 2 processes to handle the stdin and stdout getting you the progress of your operation.
First, install it:

```
sudo apt-get install pv
```

And use it like this:

```
dd if=/dev/sdX bs=1M | pv | dd of=/dev/sdY
```

You will get a progress output, look!

```
Output:
1,74MB 0:00:09 [198kB/s] [<=>]
```

Now you no longer have to guess if your dd is going or not and how much was already copied.

THE FORCE
IS STRONG WITH TRACEROUTE

by Bruno Doiche

Nothing to do on your Terminal? Well, then have some ultra nerdy fun running the following command:

```
traceroute 216.81.59.173
```

Wait a little hops, and you will see the text of a strangely familiar movie's opening crawl .

```
fs
post-fs
post-fs-data
charger
early-boot
boot
```

The core function of the init application is to start different Android services that are needed to run the complete Android framework stack. Let's take a look at few of the services included inside the init.rc.

```
service servicemanager /system/bin/servicemanager
    class core
    user system
    group system
    critical
    onrestart restart zygote
    onrestart restart media
    onrestart restart surfaceflinger
    onrestart restart drm
```

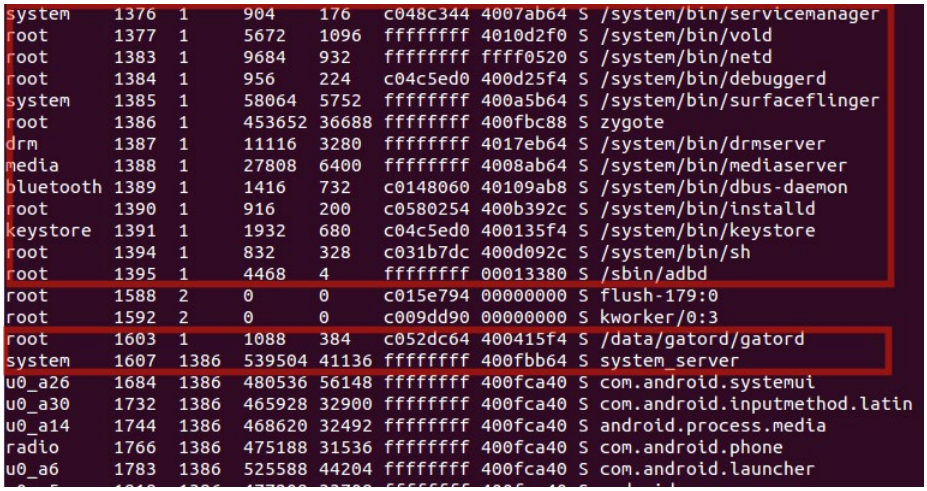
The above service command starts the servicemanager service. This application, and other binaries, reside inside Android in the /system/bin directory, and keeps track of the different services that Android starts up during the init process. The parameters below the service command specify the characteristics of servicemanager.

There is one particular line that I want to highlight here: onrestart restart zygote. The onrestart command indicates that when the servicemanager application gets restarted, it will also need to restart the zygote service. This is important is because zygote is the key application necessary for your Java layer application to run, so if the servicemanager failed to start or get restarted, the running application will also be shut down.

The service command in Android is the preferred method of launching system services during the boot process, so when you run the ps command from the Android shell (using adb shell) you will see the output shown in Figure 6.

If you open up the init.rc and cross check the service command, you will see most of the services that are needed are running in the system. The init.rc is the

Figure 6 : Application run from service command



```
system    1376  1     904    176   c048c344 4007ab64 S /system/bin/servicemanager
root      1377  1     5672   1096  ffffffff 4010d2f0 S /system/bin/vold
root      1383  1     9684   932   ffffffff ffff0520 S /system/bin/netd
root      1384  1     956    224   c04c5ed0 400d25f4 S /system/bin/debuggerd
system    1385  1     58064  5752  ffffffff 400a5b64 S /system/bin/surfaceflinger
root      1386  1     453652 36688 ffffffff 400fbc88 S zygote
drm       1387  1     11116  3280  ffffffff 4017eb64 S /system/bin/drmserver
media     1388  1     27808  6400  ffffffff 4008ab64 S /system/bin/mediaserver
bluetooth 1389  1     1416   732   c0148060 40109ab8 S /system/bin/dbus-daemon
root      1390  1     916    200   c0580254 400b392c S /system/bin/installd
keystore  1391  1     1932   680   c04c5ed0 400135f4 S /system/bin/keystore
root      1394  1     832    328   c031b7dc 400d092c S /system/bin/sh
root      1395  1     4468   4     ffffffff 00013380 S /sbin/adbd
root      1588  2     0      0     c015e794 00000000 S flush-179:0
root      1592  2     0      0     c009dd90 00000000 S kworker/0:3
root      1603  1     1088   384   c052dc64 400415f4 S /data/gatord/gatord
system    1607  1386  539504 41136 ffffffff 400fbb64 S system_server
u0_a26    1684  1386  480536 56148 ffffffff 400fca40 S com.android.systemui
u0_a30    1732  1386  465928 32900 ffffffff 400fca40 S com.android.inputmethod.latin
u0_a14    1744  1386  468620 32492 ffffffff 400fca40 S android.process.media
radio     1766  1386  475188 31536 ffffffff 400fca40 S com.android.phone
u0_a6     1783  1386  525588 44204 ffffffff 400fca40 S com.android.launcher
```

place where all the critical services need to be defined, along with the dependencies that each service depends on, including the characteristics (user, group, onrestart, etc) of each service. Failure to run any of the defined services will result in the non-functioning of Android and any relevant user applications.

Zygote

As we have seen in the previous section, Android starts a number of services that it depends on, including Zygote. It's important to note that zygote is a name of the service that is given in Android for an application that takes care of "running" user applications through the Dalvik virtual machine, as can be seen from the service shown below:

```
   service zygote /system/bin/app_process -Xzygote /system/
bin --zygote --start-system-server
       class main
       socket zygote stream 660 root system
          onrestart  write  /sys/android_power/request_state
wake
       onrestart write /sys/power/state on
       onrestart restart media
       onrestart restart netd
```

When the service starts up, it will create a local socket that is used by the internal framework to launch applications. In summary, zygote is a very thin socket-based layer that takes care of executing user application. All Android applications that you use on your device (phones, tablets, etc) are all "launched" via zygote, so if zygote is not operating, your application will not be able to launch in Android.

Let's see what will happen if you shutdown zygote from the command line. Use 'adb shell' to connect to your ODROID-U3 and execute the command stop zygote. You will immediately see the entire Android stack shut down. To start zygote again, just type start zygote.

System Services

This is the final step in the boot process, and is also essential to making life easy for developers. These services are a mixture of native and Java code that exist to fulfill the needs of user applications and services such as USB, Accelerometer, Wifi and more. When writing an Android application, you will inevitably come across these services and use them either directly or indirectly.

```
root@android:/ # service list
Found 70 services:
0       sip: [android.net.sip.ISipService]
1       phone: [com.android.internal.telephony.ITelephony]
2       iphonesubinfo: [com.android.internal.telephony.IPhoneSubInfo]
3       simphonebook: [com.android.internal.telephony.IIccPhoneBook]
4       isms: [com.android.internal.telephony.ISms]
5       commontime_management: []
6       samplingprofiler: []
7       diskstats: []
8       appwidget: [com.android.internal.appwidget.IAppWidgetService]
9       backup: [android.app.backup.IBackupManager]
10      uimode: [android.app.IUiModeManager]
11      serial: [android.hardware.ISerialManager]
12      usb: [android.hardware.usb.IUsbManager]
13      audio: [android.media.IAudioService]
14      wallpaper: [android.app.IWallpaperManager]
15      dropbox: [com.android.internal.os.IDropBoxManagerService]
16      search: [android.app.ISearchManager]
17      country_detector: [android.location.ICountryDetector]
18      location: [android.location.ILocationManager]
19      devicestoragemonitor: []
20      notification: [android.app.INotificationManager]
21      updatelock: [android.os.IUpdateLock]
22      throttle: [android.net.IThrottleManager]
23      servicediscovery: [android.net.nsd.INsdManager]
24      connectivity: [android.net.IConnectivityManager]
25      ethernet: [android.net.ethernet.IEthernetManager]
26      wifi: [android.net.wifi.IWifiManager]
27      wifip2p: [android.net.wifi.p2p.IWifiP2pManager]
28      netpolicy: [android.net.INetworkPolicyManager]
```

Figure 6 : Application run from service command

Without services, it would take a long time and effort to write Android applications. Imagine a project where you wanted to write a USB-based application, but there is no service available. You would need to write a lot of code both in Java and the native layer for your application to have access to the USB ports. You can view the currently available services by using the command service list from the ADB shell.

The class that takes care of the successful running of the services resides at frameworks/base/services/java/com/android/server/SystemServer.java. If you have a hardware project that needs to provide services to developers, it's better to have it running as an Android service so that the client application code doesn't need to be rewritten if the interface requires updates or changes.

Nanik Tolaram Lives in Sydney with his wife and 2 boys. His day job is wrestling with Android source code - customizing, troubleshooting and enhancing it to make sure it works in the hardware of choice (ARM and x86). His hobbies include breeding fish, teaching Android and electronics to other people and making stuff out of wood. He also runs the Android websites `www.ozandroid.info` *and* `kernel.ozandroid.info`

HOW TO COMPILE DOOM ON YOUR ODROID

PLAY THIS TIMELESS CLASSIC CUSTOM COMPILED FOR YOUR MACHINE

by Tobias Schaaf

There, now you even have a cover to go with your fresh build of DOOM on your ODROID

I have compiled and published many games for the ODROID, and often receive user requests for information on how to do the same for their favorite games. As an example of compiling your own games and applications, I present a comprehensive guide to compiling id Software's Doom for the ODROID U or X series.

To begin, you should have at least two copies of ODROID GameStation Turbo, available for download from the ODROID forums, burnt to SD cards. Although it's not completely necessary to have two images, you will probably want to reuse what you've already done in case you have to reinstall your OS. Also, although you have everything you need to run a certain game on the image with which you compiled it, you will need another "stock" image to test the installation script in order to make sure all of the necessary libraries are included in the final package.

I don't recommend compiling on an eMMC, since it will greatly reduce the lifetime of your eMMC module over time. I have already ruined 3 or 4 SD cards and a USB Stick by using them for heavy compiling, and I've switched over to a standard 1TB HDD instead.

WGET

Wget is a rather easy tool to use. It simply downloads single files from the Internet by supplying the URL as a command argument. In this case, we give it the link address of the SDL version of Doom:

```
wget http://www.libsdl.org/projects/doom/src/\
sdldoom-1.10.tar.gz
```

This command downloads the file sdldoom-1.10.tar.gz into the current directory. Just remember you won't be able to download a folder or multiple files with wget - it's just for single files.

Setting up an build environment

Here is a list of recommended programs that should be installed before the compilation begins:

```
apt-get install build-essential cmake automake autoconf
git subversion checkinstall
```

You will probably need a lot more applications than those listed, but it is a good start. Next, create a dedicated folder in which to build your binaries.

```
mkdir sources
```

Oh, the memories of being a teenager struggling to kill cyberdemons! For a long time, I thought he was the final boss of the game because I died so many times there.

This folder can be located on an external device like a USB stick or HDD as well, which makes it easier to use on a different image, a different ODROID, or even on another PC.

Start with an easy build

To begin building the Doom SDL application, type the following:

```
cd sources
wget http://www.libsdl.org/projects/doom/src/sdldoom-1.10.tar.gz
tar xzvf wget sdldoom-1.10.tar.gz
cd sdldoom-1.10
```

In the sdldoom-1.10 folder, you'll find a very important file called "configure", which many programs include in their source code.

Configure

Configure is a program that scans your system and checks your build environment for the necessary pieces required for compiling the program. It normally informs you about missing files, and often allows you to add special parameters to tweak your build. To see what parameters are offered by a certain program you want to compile you can start configure with the --help parameter.

```
./configure --help
```

There are a lot of parameters, but don't get confused, since most of them are not needed for most standard builds. By default, typing make install will copy all of the compiled files to the system directories such as /usr/local/bin and /usr/local/lib. If you wish to change the installation directory, you can specify an installation prefix other than /usr/local using the configure --prefix option, for example, ./configure --prefix=$HOME.

Compiling the sdl-doom source code

Normally, there is a README file which tells you what dependencies are needed. This version of Doom does not actually have such a README file. However, the configure command will highlight any missing dependencies. In the following example, the library called "libsdl1.2-dev" is missing:

```
./configure
loading cache ./config.cache
checking for a BSD compatible install... (cached) /usr/bin/install -c
[...]
checking for sdl-config... (cached) /usr/bin/sdl-config
checking for SDL - version >= 1.0.1... ./configure: 1: ./configure: /usr/bin/sdl-config: not found
./configure: 1: ./configure: /usr/bin/sdl-config: not found
no
*** Could not run SDL test program, checking why...
*** The test program failed to compile or link. See the file config.log for the
*** exact error that occured. This usually means SDL was incorrectly installed
*** or that you have moved SDL since it was installed. In the latter case, you
*** may want to edit the sdl-config script: /usr/bin/sdl-config
configure: error: *** SDL version 1.0.1 not found!
```

This is where the compilation process gets a little bit cryp-

tic and you will need some experience to figure certain stuff out. In this case, configure is reporting that it could not find sdl-config and also complains that SDL version 1.0.1 was not found. However, since the source code messages are meant to be generic, it does not tell you EXACTLY want you need to install but rather gives you a name you need to work with.

In this case, it tells you the program is called SDL but we already know the required file in fact is a program called "libsdl1.2-dev". In most cases, you will need some experience to figure out what is needed, but there are a couple of tools that might help you figure out what you need.

```
apt-cache search
```

With the command apt-cache search you can try to find certain packages that you do not know the full name of.

```
apt-cache search SDL
```

Depending on your search words, these lists can be rather long, since it's searching on files name as well as descriptions. It's better to search for something that might not be too common, such as the term "libsdl":

```
apt-cache search libsdl
```

By using the "lib" prefix, the list is shorter and shows us that there are actually a few libraries that start with libsdl. It's important to remember that when compiling a package, the dependent libraries are always development libraries, containing the the header files. So when running the apt-search command, we are only interested in the libraries ending with "-dev". We can just search for these dev libraries by adding another keyword to our search:

```
apt-cache search libsdl dev
```

With the additional search terms, the resulting list is nice and short, and shows us that there is a libsdl1.2-dev library that matches the version mentioned by the configure command, which requires version 1.0.1 or above. We can then install it with the following:

```
apt-get install libsdl1.2-dev
```

Now that the libsdl1.2-dev package is installed, let's try configure again:

```
  ./configure
  loading cache ./config.cache
  checking for a BSD compatible install...
(cached) /usr/bin/install -c
  checking whether build environment is
sane... yes
  checking whether make sets ${MAKE}...
(cached) yes
  checking for working aclocal... found
  checking for working autoconf... found
  checking for working automake... found
  checking for working autoheader... found
  checking for working makeinfo... missing
  checking whether make sets ${MAKE}...
(cached) yes
  checking for gcc... (cached) gcc-4.7
  checking whether the C compiler (gcc-4.7
) works... yes
  checking whether the C compiler (gcc-4.7
) is a cross-compiler... no
  checking whether we are using GNU C...
(cached) yes
  checking whether gcc-4.7 accepts -g...
(cached) yes
  checking for a BSD compatible install...
/usr/bin/install -c
  checking for sdl-config... (cached) /usr/
bin/sdl-config
  checking for SDL - version >= 1.0.1...
yes
  creating ./config.status
  creating Makefile
```

Grinning faces, guns blazing — this is what makes this one of the best games ever! Playing all night... I regret nothing!

Now that no errors are being reported, we are ready to go! Notice the last line: "creating Makefile". A Makefile is always something that you need nearly all of the time. If you have a Makefile in your folder you can simply type "make" and it will start compiling your program automatically. You can also add the parameter make -j4 which will use 4 threads to create a program, increasing the build speed since it uses all 4 cores of the ODROID.

Advanced information about configure

Sometimes configure will instead complain about something like "-lSDL_mixer" missing. The leading "-l" gives you the hint this is a library, and the rest tells you to look for something with mixer.

```
apt-cache search libsdl mixer dev
libsdl-mixer1.2-dev - Mixer library for Simple DirectMedia Layer 1.2, development files
```

Other times, the configuration step might complain about some missing header files (files ending with .h) like "SDL/SDL_net.h". There are certain sites on the net for Debian and Ubuntu, where you can search for files within certain packages and find out what the names of the missing packages are:

Ubuntu: `http://bit.ly/PSihOu`
Debian: `http://bit.ly/1rQEbzW`

Completing the build

You now should have a fully running program called "doom" in your folder and can run Doom from here if you give it the .wad file from your original Doom disks. All done, right?

Well, not entirely. Doom is done building and you can play it by copying your .wad files into the same folder, but it's not yet "installed". Most programs allow you to do a simply type make install which will copy all necessary files in the right folder to complete the installation.

```
make install
make[1]: Entering directory `/home/odroid/sources/sdldoom-1.10'
/bin/sh ./mkinstalldirs /usr/local/bin
  /usr/bin/install -c doom /usr/local/bin/doom
make[1]: Nothing to be done for `install-data-am'.
make[1]: Leaving directory `/home/odroid/sources/sdldoom-1.10'
```

With doomsdl it's only the binary file (doom) that's getting installed, but depending on what you're building, the installation can be a lot bigger and install thousands of files.

So how do you install it to another system? You could copy the files manually, but if it's a large number of files, this can be time-consuming, and may not include all of the dependencies and libraries required to run the application. This is where checkinstall comes in handy, which creates portable .deb files which you can install on another system. In next month's article, I'll cover how to use checkinstall to copy Doom to a fresh installation of Linux.

RECOMPILE THE MALI VIDEO DRIVERS

FIXING GRAPHICS ISSUES WHEN UPGRADING TO UBUNTU 14.04

by Rob Roy

When upgrading to Ubuntu 14.04, it's likely that the current version of Xorg Server 1.14, the software responsible for creating the graphical user interface, is no longer compatible with the Mali video drivers that worked with Ubuntu 13. This can cause slow windows and blank screen issues. To compile the latest version of the Mali software, type the following into a Terminal window or an SSH session:

```
$ wget http://malideveloper.arm.com/downloads/drivers/DX910/r3p2-01rel4/DX910-SW-99003-r3p2-01rel4.tgz
$ tar xzvf \
  DX910-SW-99003-r3p2-01rel4.tgz
$ cd DX910-SW-99003-r3p2-01rel4/x11/
  xf86-video-mali-0.0.1
$ sudo apt-get install autoconf \
  xutils-dev libtool \
  xserver-xorg-dev
$ cd src
$ rm compat-api.h
$ wget http://cgit.freedesktop.org/ \
  ~cooperyuan/compat-api/plain/ \
  compat-api.h
$ cd ..
$ ./autogen.sh
$ ./configure --prefix=/usr
$ make
$ sudo make install
$ cd /etc/X11/xorg.conf.d
$ sudo mv \
  exynos.conf \
  exynos.conf.original
$ sudo reboot
```

The most popular version of the Mali 400 frame buffer drivers as of April 2014 is r3p2-01rel4, and you can check for an updated release of the Mali software by visiting the developer site at `http://bit.ly/1f5Jk51`.

After rebooting, the Xorg Server version (14) will match the Mali driver version, and the HDMI signal should be working properly. If the HDMI signal is working, but the desktop windows are moving slowly, typing the lines below restore the Mali drivers upon reboot. The Mali configuration is stored in `/etc/X11/xorg.conf`, and renaming the ARM SoC configuration file prevents it from overriding the Mali version.

2 SYSTEMS, 1 ODROID, PURE FUN!

HOW TO MAKE A DUAL BOOT SYSTEM WITH ANDROID AND UBUNTU

by Yong-Oh Kim, Hardkernel Developer

Ubuntu and Android are two popular operating systems available for the ODROID, and can be combined to run together in several ways. You can create a multi-boxed system by installing each OS on a separate ODROID and connecting them using communication protocols, run a headless version of Linux inside Android using the chroot command, connect them via USB cable for fast file transfers, or create a client-server relationship using a web server and development environment. This article outlines yet another method of combining Ubuntu and Android: By creating a flexible dual boot system that allows either OS to be run from the same hard drive.

The Android operating system offers many consumer-oriented applications like Netflix, Hulu, Skype, Google Hangout, modern 3D/2D games, 1080p capable XBMC, and other apps aimed at entertainment and social interaction. On the other hand, Ubuntu OS offers a PC-like experience, with many productivity applications and developers' utilities like LibreOffice, GIMP, Apache server, Eclipse, Arduino IDE, OpenMP/CV libraries, C/C++/JAVA/Python programming tools, and many other technical applications.

However, the file systems in both Android and Ubuntu are very different from each other, and the user can waste a lot of storage as well as not be able to share content easily between the two systems. There are 5 main steps in building a dual boot OS image:

Modify the Android source code for the MTP and build the platform source.
Grab the Xubuntu root file system and modify the odroid-config for new partition table.
Create new partition table and format partitions in your eMMC storage.
Copy Xubuntu root file system and change the UUID.
Transfer Android root file system via fastboot protocol.

For best performance, an eMMC module or very fast (20MB/s) SD card is recommended, as the Android operating system will perform poorly when using a slow SD card.

Modify the Android source code for the MTP

Download the odroidu.zip file from `http://bit.ly/PXpjkR`, then unzip and overwrite it into the device/hardkernel/odroidu/ directory of the Android platform source code. It's also necessary to modify the package/app/Utility/src/com/hardkernel/odroid/MainActivity.java file to access the MTP in Android instead of VFAT, as shown below.

```
$ svn diff packages/apps/Utility/
Index:    packages/apps/Utility/src/com/
hardkernel/odroid/MainActivity.java
@@ -20,6 +20,7 @@
import android.app.Activity;
import android.content.Context;
import  android.content.SharedPreferenc-
es;
+import android.os.Environment;
import android.os.Bundle;
import android.util.Log;
import android.view.Menu;
@@ -44,7 +45,7 @@
 public final static String MIN_FREQ_NODE
=    "/sys/devices/system/cpu/cpu0/cpufreq/
scaling_min_freq";
  //private final static String BOOT_INI
= "/storage/sdcard1/boot.ini"; //"/mnt/sd-
card/boot.ini";
- private final static String BOOT_INI =
"/mnt/sdcard/boot.ini";
```

```
  + private String BOOT_INI = "/mnt/sdcard/
boot.ini";
   public int mCurrentMaxFreq;
   public int mCurrentMinFreq;
  @@ -371,6 +372,14 @@
     tv.setVisibility(View.GONE);
    }

  +        File sdcard1 = new File("/stor-
age/sdcard1");
  +        if (sdcard1.exists()) {
  +            Log.e(TAG, "MTP");
  +          BOOT_INI = "/storage/sdcard1/
boot.ini";
  +        } else {
  +            Log.e(TAG, "Mass Storage");
  +        }
  +
    File boot_ini = new File(BOOT_INI);
    if (boot_ini.exists()) {
     try {
```

After building the full Android source code, you will have the system.img which packages the entire Android root file system. If you don't want to build the full Android source code, you can use a prebuilt image, available for download from `http://bit.ly/1i5ZJB3` or `http://bit.ly/1rWMMB9`.

Copy the Xubuntu root file system

A Linux host PC should be used to store and transfer the files needed for dual boot.

```
  mkdir boot
  sudo  cp  -a  /media/codewalker/BOOT/*
boot/
  mkdir rootfs
  sudo  cp  -a  /media/codewalker/rootfs/*
rootfs/
```

Modify the odroid-config script

Update the script located at rootfs/usr/local/sbin/odroid-config and expand the file system functions by changing "mmcblk0p2" to "mmcblk0p3".

```
  40 do_expand_rootfs() {
  42    p2_start=`fdisk -l /dev/mmcblk0 |
grep mmcblk0p3 | awk '{print $2}'`
  43   fdisk /dev/mmcblk0 <<EOF
  44 p
  45 d
  46 3
  47 n
  48 p
  49 3
  50 $p2_start

  72 case "$1" in
  73   start)
  74    log_daemon_msg "Starting resize2fs_
once" &&
  75     resize2fs /dev/mmcblk0p3 &&
  76     rm /etc/init.d/resize2fs_once &&
  77     update-rc.d resize2fs_once remove
&&
  78     log_end_msg $?
  79     ;;
  80   *)
```

Check the u-boot version

You should use the latest u-boot (Jan 12 2014 or later), otherwise the fdisk command will not work properly. To do so, install the Android release 2.6 onto your eMMC from `http://bit.ly/PXriWq`. Then, check for the version by using the USB-UART serial console (cable sold separately), which should look like this:

```
  OK

  U-Boot 2010.12-svn (Jan 27 2014 - 15:07:10)
for Exynox4412

  CPU: S5PC220 [Samsung SOC on SMP Platform
Base on ARM CortexA9]
  APLL = 1000MHz, MPLL = 880MHz
  DRAM:  2 GiB
```

Generate a new partition table

Using the USB-UART console, enter into the u-boot prompt and use the fdisk command to create the partition table. The format of the command is fdisk -c [boot device:0] [system] [userdata] [cache] [vfat]

```
  Exynos4412 #  fdisk -c 0 512 -1 128 300
  Count: 10000
  NAME: S5P_MSHC4
  fdisk is completed

  partion #     size(MB)      block start #
block count    partition_Id
     1              306            1462846
626934          0x0C
```

```
     2              517              134343
1059817          0x83
     3             6362             2089780
13031271         0x83
     4              131             1194160
268686           0x83
```

Item	Description	File System	Size
Boot zImage ramdisk	BL1/BL2/U-boot/ENV Android Kernel Android Ramdisk(not used)	RAW	66MB
SYSTEM (mmcblk0p2)	Android System Gapps size is considered	EXT4	512MB
CACHE (mmcblk0p4)	Android Cache	EXT4	128M
BOOT (mmcblk0p1)	Linux Kernel boot.scr or boot.ini	FAT16	300M
USERDATA (mmcblk0p3)	Android Userdata Ubuntu rootfs	EXT4	All the rest

The chart below outlines the different parts of the partition table, with each part color-coded for the operating system that uses it (Common, Android, Ubuntu)

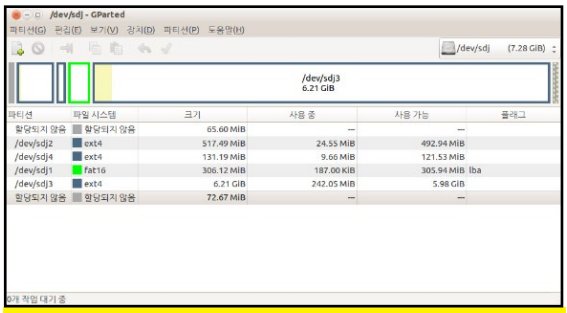
Format the partitions

To prepare the partitions, power down the ODROID and remove the eMMC module. Then, plug it into any Linux host using the adapter that came with the eMMC and a USB SD Card adapter. Note that the partition table entries listed are in logical order instead of the physical order that they appear on disk.

```
  $ sudo fdisk -l
  Disk /dev/sdX: 7818 MB, 7818182656 bytes
  253 heads, 59 sectors/track, 1022 cylin-
ders, total 15269888 sectors
  Units = sectors of 1 * 512 = 512 bytes
  Sector size (logical/physical): 512 bytes
/ 512 bytes
  I/O size (minimum/optimal): 512 bytes /
512 bytes
  Disk identifier: 0x00000000

    Device Boot      Start         End
Blocks   Id  System
  /dev/sdX1            1462846      2089779
313467    c  W95 FAT32 (LBA)
  /dev/sdX2             134343      1194159
529908+  83  Linux
  /dev/sdX3            2089780     15121050
6515635+  83  Linux
  /dev/sdX4            1194160      1462845
134343   83  Linux
```

Next, format the eMMC in preparation for the Ubuntu and Android root file systems.

```
  [~]$ sudo mkfs.vfat /dev/sdX1
  mkfs.vfat 3.0.16 (01 Mar 2013)
  [~]$ sudo mkfs.ext4 /dev/sdX2
  mke2fs 1.42.8 (20-Jun-2013)
  Filesystem label=
  OS type: Linux
  Block size=4096 (log=2)
  ...

  [~]$ sudo mkfs.ext4 /dev/sdX3
  mke2fs 1.42.8 (20-Jun-2013)
  Filesystem label=
  OS type: Linux
  Block size=4096 (log=2)
  ...

  [~]$ sudo mkfs.ext4 /dev/sdX4
  mke2fs 1.42.8 (20-Jun-2013)
  Filesystem label=
  OS type: Linux
  Block size=1024 (log=0)
  ...
```



Check the partitions with GParted

Install the Ubuntu rootfs

```
  $ sudo cp –a boot/* /media/[user]/[mount_
point]/
  $ sudo cp –a rootfs/* /media/[user]/
[mount_point]/
```

The [user]/[mount_point] corresponds to the directory where the eMMC adapter is mounted on the Linux host. The boot partition corresponds to /dev/sdX1 (FAT), and

the rootfs partition corresponds to the /dev/sdX3 partition (EXT4 userdata).

Replace UUID in boot.scr

Inspect the boot.scr file in the "boot" partition and update the /dev/sdX3 partition so that the UUID matches the one listed in boot.scr:

```
  $ cat /media/codewalker/5145-2E60/boot.
scr

                              '^E^YVOÚ<9f>7R}-
>^@^@^A<^@^@^@^@^@^@^@^@^E^?ß9^E^
B^F^@boot.scr for X with HDMI auto-
pr^@^@^A4^@^@^@^@setenv initrd_high
"0xffffff"
  setenv fdt_high "0xffffff"
  setenv bootcmd "fatload mmc 0:1 0x40008000
zImage; fatload mmc 0:1 0x42000000 uInitrd;
bootm 0x40008000 0x42000000"
  setenv bootargs "console=tty1
console=ttySAC1,115200n8
root=UUID=e139ce78-9841-40fe-8823-
96a304a09859 rootwait ro mem=2047M"
   boot

  $ sudo tune2fs /dev/sdX3 -U e139ce78-
9841-40fe-8823-96a304a09859
  tune2fs 1.42.8 (20-Jun-2013)
```

Install the Android system files

Since the original image already contains the Android files, the Android partitions can be populated by entering the u-boot prompt and typing the following fastboot commands:

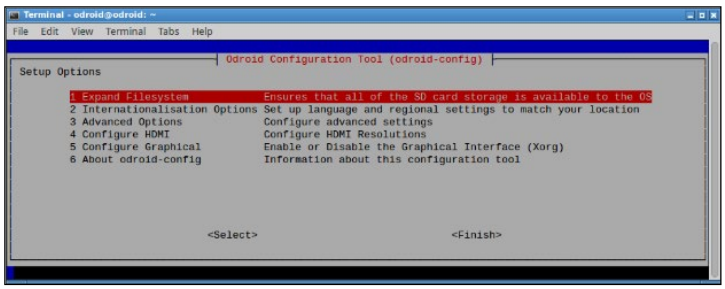
```
# fastboot
$ fastboot flash system system.img
$ fastboot reboot
```

Now that both Ubuntu and Android are installed, the eMMC module is ready to use!

Using the prebuilt image

If you are not an Android platform developer or want to use the dual boot system right away, we provide a prebuilt image, which can be downloaded from `http://bit.ly/1i11nII` or `http://bit.ly/1i6bSWQ` and flashed onto your eMMC.

The prebuilt image uses Xubuntu as its default OS. Just expand the file system and restart the OS on first boot. The expanded Ubuntu root file partition will be accessible from Android as a data partition.



Odroid Configuration Tool, nothing says clean build and install as running this chap!

Switching between Android and Ubuntu

When you want to change the default boot option to Android, open a terminal window in Xubuntu and run the bootA.sh script using super user permissions by typing sudo /media/boot/bootA.sh. If you created your own dual boot system, you should add the bootA.sh script manually:

```
#!/bin/sh
cd /media/boot
mv boot.ini.android boot.ini
mv boot.scr boot.scr.ubuntu
sync
reboot
```

If you want to change the default boot option back to Ubuntu from Android, open the Android Terminal app, and run the bootL.sh script, which will also require superuser permissions:

```
# su
# sh /storage/sdcard1/bootL.sh
```

For manually created systems, the bootL.sh script to switch back to Xubuntu should contain the following:

```
#!/bin/sh
cd /storage/sdcard1
mv boot.scr.ubuntu boot.scr
mv boot.ini boot.ini.android
sync
reboot
```

For a demonstration of the dual boot system, please refer to our instructional video at `http://youtu.be/osERBvQN5ME`.

GETTING STARTED WITH YOUR ODROID

HOW TO COPY AN IMAGE FILE TO AN SD CARD OR EMMC

by Venkat Bommakanti

Many official and community prebuilt images are available for download from the ODROID forums at `http://forum.odroid.com`, but it is sometimes difficult for new ODROID owners to learn how to use the images to create a bootable disk. This article outlines the process of downloading, verifying and installing an .img or .img.xz file using a Linux, Mac OSX or Windows host.

General Requirements

1. Any ODROID computer, with an appropriate power adapter
2. A MicroSD card (with an SD card reader/writer) or an 8+ GB eMMC
3. A downloaded image file whose filename ends in either .img or .img.xz

Obtain the image and checksum files

To download the image file, first create a working folder in which to place the image on a Linux, OSX or Windows host computer. For instance, If you intend to use a prebuilt official Ubuntu Hardkernel image, the compressed .img.xz file(s) can be downloaded from `http://bit.ly/1iPCvzf`. Note that any U2 image would also work with the U3 board (and vice versa). To ensure file integrity, also make sure to download the corresponding checksum (.xz.md5sum) file from the same location. For this example, the set of downloaded files used in this article is:

```
xubuntu-13.10-desktop-armhf_odroidu_20140211.img.xz
xubuntu-13.10-desktop-armhf_odroidu_20140211.img.xz.md5sum
```

If you intend to use an image from elsewhere, note that you would need to ensure the authenticity of the image and that it is safe to be used. Download the image files only from a trusted source such as the ODROID forums, community repository, or the Hardkernel site.

Linux

Please note that the procedure listed here uses the Linux disk-duplicate (dd) command. As with numerous Linux commands, it needs to be used with proper care - if not, you may inadvertently render the host Linux system useless, as critical disk partitions have the potential to be overwritten. Some of the parameters in the commands listed here may need to be altered to use information specific to your setup.

In a Terminal window, navigate to the folder where you downloaded the image using the cd command. Then, evaluate the md5sum for the downloaded image file by typing:

```
md5sum xubuntu-13.10-desktop-armhf_odroidu_20140211.img.xz
```

Compare the result with the contents of the md5sum file obtained from the server. In this particular case, the md5sum to be used for matching would be:

```
605ac6805feb2867d78c45dd660acc80
```

If they match, the file integrity is ensured and one can proceed to the next step. If not, the image file may have been corrupted and should be re-downloaded. A mismatch in the md5sum may imply an altered or corrupt image, especially possible when the authenticity of the download website is questionable.

Once the md5sum has been verified to match, unpack the compressed image using the xz command:

```
xz -d xubuntu-13.10-desktop-armhf_odroidu_20140211.img.xz
```

This will replace the compressed file with an image file ending in .img. The next step is to determine what label the Linux host has given to the SD or eMMC module to which the image will be written. In the already running Terminal window, without inserting the SD card, run the df -h command and note the output reflecting various mounted drives. The output may be something like this:

```
$ df -h
Filesystem    Size   Used    Avail  Use%  Mounted on
/dev/sda1     46G    3.4G    40G    8%    /
none          4.0K   0       4.0K   0%    /sys/fs/cgroup
udev          2.0G   4.0K    2.0G   1%    /dev
tmpfs         396M   880K    395M   1%    /run
none          5.0M   0       5.0M   0%    /run/lock
none          2.0G   152K    2.0G   1%    /run/shm
none          100M   76K     100M   1%    /run/user
```

Note that in this case, /dev/sda1 reflects the filesystem corresponding to the first partition of the first storage device, which in this example is 50 GB.

Now, insert the SD card and rerun the same command:

```
$ df -h
Filesystem  Size  Used  Avail Use%  Mounted on
/dev/sda1   46G   3.4G  40G   8%    /
none        4.0K  0     4.0K  0%    /sys/fs/cgroup
udev        2.0G  4.0K  2.0G  1%    /dev
tmpfs       396M  880K  395M  1%    /run
none        5.0M  0     5.0M  0%    /run/lock
none        2.0G  152K  2.0G  1%    /run/shm
none        100M  76K   100M  1%    /run/user
/dev/sdb1   15G   32K   15G   1%    /media/terrapin/XFER
```

Note that in this case, /dev/sdb1 is the only new entry, and will reflect that the target SD card has been inserted and that the first partition has been mounted. Although the partition number is appended to the name, the SD card raw disk device actually has the name /dev/sdb (without the number 1). If using an eMMC, the system may assign a label such as /dev/mmcblk0 instead, and the first partition will be mounted as /dev/mmcblk0p1.

If your SD card is new and unformatted, you can skip the next step, which is to clear out the SD card by writing zeros to it. Zeroing the card assures that no previously leftover data is present on the disk, which may disrupt the new partition scheme as well as making any backup copies larger than necessary when compressed.

In this example, we have seen that the SD card contains one formatted partition. It needs to be unmounted first, using the umount command, substituting the disk label specific to your scenario:

```
sudo umount /dev/sdb1
```

Then, zero out the above partition using the command:

```
sudo dd if=/dev/zero bs=4M of=/dev/sdb
&& sync
```

It is worth reiterating that, since partitions are being deleted and the SD card is being formatted, one should exercise extreme caution. If unsure of the usage of these powerful commands, it may be safer to create a Linux Virtual Machine (VM) using Oracle's VirtualBox (`https://www.virtualbox.org/`) and then execute the commands from within the VM. In the worst case scenario, the VM instance would get ruined, leaving the actual host Linux system intact.

Wait for completion of the formatting process before proceeding to the next step, which may take anywhere from 15 minutes to 2 hours, depending on the speed of the SD card or eMMC module. Once completed, the Terminal window will report that the card is out of disk space (which is normal), indicating that the zeros have been written successfully to disk.

From the folder that has the extracted image, write the image to the formatted SD card using the raw disk device name as follows:

```
sudo dd bs=4m if=xubuntu-13.10-desktop-
armhf_odroidu_20140211.img \
of=/dev/sdb
```

The device name needs to be specified carefully in this command as noted earlier, leaving off any integers, which correspond to the individual partitions rather than the entire disk.

This write process will again take a while (up to 2 hours) to complete. In case of success, the output will contain the number of records (in and out), bytes copied, data copy-rate and duration of the copy. The sync command flushes data from the write-cache, ensuring that the image has been completely written to disk.

In case of failure, follow the actionable output, if any. It may be worthwhile to reformat the SD card and retry the procedure. If it fails again, it is preferable to use another SD card of similar capacity, that is known to be working properly.

Upon completion of the dd command and display of successful output, the SD card will be automatically re-mounted. Re-run the df command listed earlier, to ensure successful remount of the SD card, then eject the card using the command:

```
sudo eject /dev/sdb
```

The image is now ready for booting! If the ODROID is currently running, shut it down gracefully, then insert the SD card and power it back on. It should now start up using the new OS image, and be ready for you to enjoy.

OSX

In addition to the general requirements mentioned above, the OSX system should also have an installed copy of Unarchiver, which is a useful utility for compressing and decompressing image files, available at `http://bit.ly/1iLr5m3`. Note that Unar-

GET YOURSELF
A LITTLE MORE PERSONALITY ON YOUR SUDO

by Bruno Doiche

No one likes being insulted, of course, but sometimes your Linux looks like a souless machine when you issue a sudo su - and by mistake an incorrect password:

```
odroid@goonix:~$ sudo su -
[sudo] password for odroid:
Sorry, try again.
```

How boring is that, right? Nothing that can't be fixed by issuing:

```
sudo visudo
```

Add the following line:

```
Defaults        insults
```

And now, when you sudo a command and put the wrong password it goes politely as that:

```
odroid@goonix:~$ sudo su -
[sudo] password for odroid:
Hold it up to the light --- not a brain in sight!
[sudo] password for odroid:
You can't get the wood, you know.
[sudo] password for odroid:
There must be cure for it!
```

It's like having your own snarky art editor living in your Terminal!

ANOTHER
SUDO SECURITY TIP

A good security practice, is to never set your user to sudo automatically on a machine that another person can access. And after exiting sudo, it doesn't ask immediately for your password! Save yourself by typing the command below:

```
sudo -K
```


chiver has several Macintosh-specific versions, so make sure to download and install the OSX version appropriate for your system.

The procedure for checking the md5sum of the downloaded file is similar to Linux, but uses the command md5 instead of md5sum. A handy shortcut for inspecting the checksum is to open a Terminal window and type md5 followed by a [SPACE] character. Then, using the mouse, drag the compressed image file (*.img.xz) into the terminal window. The command line will be updated with the compressed file name. Now hit the [ENTER] key. The md5sum of the compressed image file is returned as the output. Compare the result with the contents of the md5sum file to make sure that the file has been downloaded correctly. For more information on checking the md5sum, please refer to Ubuntu's OSX md5sum help page at `http://bit.ly/1nTVz7q`.

Assume the Unarchiver Version 3.9.1 utility, which is the latest as of this article, has been installed on your Mac and is set to be the default utility to unpack compressed files. Start the program and configure the utility to:

1. Retain the original downloaded file (post unpacking)
2. Place the unpacked image file in the same folder as the location of the compressed file
3. Retain the modification date of the compressed file (to keep track of the image information)

Decompressing the file with these options should result in the creation of a file ending in .img in the same folder as the original .img.xz file.

Although df -h can also be used to check the available mounted drives, OSX provides a customized command called diskutil which can be used instead and provides more straightforward output. In the Terminal window, type the following command before inserting the SD card:

```
diskutil list
```

Note that, in OSX, mounted drives are named as /dev/diskX rather than the Linux convention of /dev/sdX. If the SD card is new, skip the next step as it is not necessary to zero out a fresh card.

To prepare the SD card or eMMC module, start the OSX Disk Utility application and click on the target SD card on the left of the window. Press the "Security Options" button at the bottom center, and select the "Zero Out Data" option in the popup window. Press OK, then click the "Erase" button and wait until the progress bar reaches 100%. Once the disk has been zeroed, it is ready to accept the new image.

Because OSX auto-mounts any pluggable media, the drive must be first unmounted by using the command:

```
sudo diskutil unmountdisk /dev/disk2
```

Then, write the image to the SD card using the raw disk device name in the dd command. Note the lowercase "1m" which differs from the uppercase Linux syntax:

```
sudo dd bs=1m if=odroidu2_20130125-linaro-ubuntu-desktop_SDeMMC.img of=/dev/disk2
```

The raw disk device name needs to be specified carefully in this command as noted earlier. Wait for the completion of the command to be notified of success or failure.

Once the dd command successfully completes, the SD card will again be automatically re-mounted. You can eject the card using the following command:

```
sudo diskutil unmountdisk /dev/disk2
```

Wait until the disk's icon disappears from the desktop, remove the SD card or eMMC module from the Macintosh, insert into the ODROID and apply power to begin using the new operating system.

Windows

Windows does not natively support the Linux ext3/ext4 partition type, so several additional utilities are required in order to copy an image file to disk:

1. 7-Zip (`http://www.7-zip.org`) file compression utility to extract the SD card image from either the downloaded .xz file.
2. Improved Win32DiskImager (`http://bit.ly/1q1HTsW`) utility to write the .img file to your SD-Card.
3. MD5sums (`http://bit.ly/1ukeVUZ`) utility to evaluate the checksum (integrity) of a downloaded file. This is optional but useful to ensure that an image file matches the version on the server.

After downloading the .img.xz file as described above, evaluate the md5sum using the command:

```
c:\Program Files (x86)\md5sums-1.2\md5sums
xubuntu-13.10-desktop-armhf_odroidu_20140211.img.xz
```

Compare the result with the contents of the md5sum file and continue on to the next step if they match. Once the file has been verified to be intact, use the 7-Zip utility to extract the image from the compressed file:

```
c:\Program Files (x86)\7-zip-7z920 -z
xubuntu-13.10-desktop-armhf_odroidu_20140211.img.xz
```

For convenience of Windows users, Hardkernel publishes a special purpose Win32DiskImager utility which automatically writes zeros to the SD card before copying the image, so that everything can be done in a single step. When launched, it will display a similar interface to that shown in the screenshot here at left.

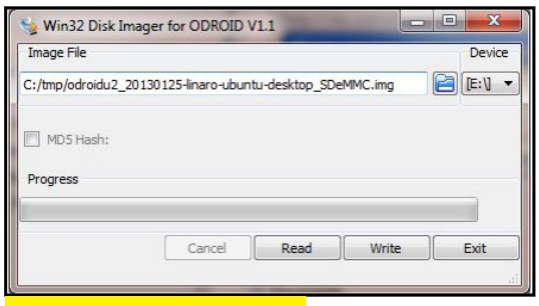


Image installation using Improved WinDiskImager

Select the desired parameters as shown, and start the image installation by clicking the "Write" button. The extra time needed to write zeroes to the image may add about 30 minutes or more to the writing process.

Finally, eject the disk by right-clicking on the SD card in File Explorer and selecting the "Eject" option. Insert the SD card into the ODROID, power it up, wait for the boot process to complete, and enjoy your new operating system.

For additional information or questions on copying image files to SD card, please refer to Osterluk's ODROID wiki at `http://bit.ly/1rQgqWH`.

SORT BY FILE SIZE
INSIDE A DIRECTORY

by Bruno Doiche

Want to know which files are the largest inside a specific directory? Type the following command in the Terminal window:

```
find . -type f -exec ls -s {} \;
| sort -n -r | head -5
```

This is useful when you have to do some housekeeping! Need to see only files greater than a certain size? Use this command, which lists all files that are larger than 100MB:

```
find ~ -size +100M
```

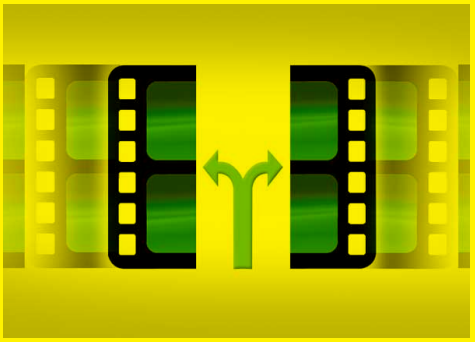
SPLIT
A HUGE FILE

Have you finally gotten ahold of that wonderful show filmed in pristine high definition at your friend's computer, but found that it is 7GB while all you have is two 4GB flash drives? Use this command:

```
split -b 1GB [yourvideofile.
mkv] [yoursplitvideofile]
```

The file will be sliced into 1GB chunks which you can then copy to your flash drives. When at home, copy them back to your hard disk and type this command to reassemble the original file:

```
cat [yoursplitvideofile*] >
[yourvideofile.mkv]
```



ON THE THERMAL BEHAVIOR OF ODROIDS

THE PERFORMANCE DIFFERENCE BETWEEN THE XU AND U3 IN GREATER DETAIL

by Jussi Opas

For comparison, two ODROIDs, the XU and U3, were tested in parallel in order to gauge their relative difference in performance, temperature, and frequency scaling behavior. We can safely assume that XU is faster than U3, but the question is, how much faster is it? So that we can have a more informed opinion than just intuition, we generated some measurement statistics between the two machines. For these test, the stock XU board has a heatsink and attached fan, while the U3 has a heat sink without a fan, as an example of a passively cooled system.

The two computers have very different specifications, as has been shown in the table below. The U3 has a quad-core ARM Cortex A9 processor, while the XU has a big.LITTLE processor having two separate process clusters: one with four A7 ARM cortex cores and another with four A15 cores. Both boards come with a 2GB PoP (package on package) memory, but the type of memory included with the XU is faster than the memory that comes with the U3. When running the official Hardkernel Ubuntu distributions, the default frequency scaling governor of the U3 is set to "performance" while the XU uses the "ondemand" setting by default. The factory-set clock frequency of the 1.7GHz U3 is 100 MHz higher than the XU's 1.6GHz frequency. The ability to overclock each board may vary, and the values given in the table are based on one unit of each. The U3 was tested with 3.8 Linux kernel, and a self-tuned 3.4 kernel was used to test the XU.

Each ODROID potentially behaves differently when the processors are fully utilized by an intensive application. Both SoCs have also a GPU, but their behavior is not of concern here, because no graphical computations are assigned to them by our test application.

We know that you do that comparsion a dozen times a day when thinking which one you are going to buy, so we did a nice little table here

	ODROID U3	ODROID XU
SoC	Exynos 4412 Prime	Exynos5 5410
CPU	ARM 4xA9	ARM 4xA15 and 4xA7
Memory	2 GB, LPDDR2	2 GB, LPDDR3
Default governor	performance	ondemand
Default max frequency	1.7 GHz	1.6 GHz
Overclockability	1.92 GHz	1.8 GHz or more
Cooling	Heat sink	Heat sink with embedded fan
Kernel	3.8.13.18	3.4.74 (customized)

All computations are made by the CPU, and RAM is not a limiting factor,
Computations are made with multiple threads using Java with a real world like application, so the application has not been written only for testing purposes,
The same test run is made with differing numbers of threads so that cores are 100% utilized, therefore we need not consider CPU utilization,
No file IO is used,
All computations consist of integer, float, and double adds, subtractions, divisions, multiplications, square roots, some Java Math methods, array access and assignments, and object creation and deletion.

We hope that the application used for testing has been hardened that it does not have internal flaws, and that we can trust the results that are shown in my previous article [OP14].

The test application, as used here, takes a computer to the edge of its capabilities, which rarely happens during normal everyday use. The trials were done at normal room temperature, which is about 22C degrees. The comparison uses the default frequencies (1.6 and 1.7 GHz).

Based on the numbers in the figure, we can conclude that the XU is about 25% faster than the U3. However, the U3 can

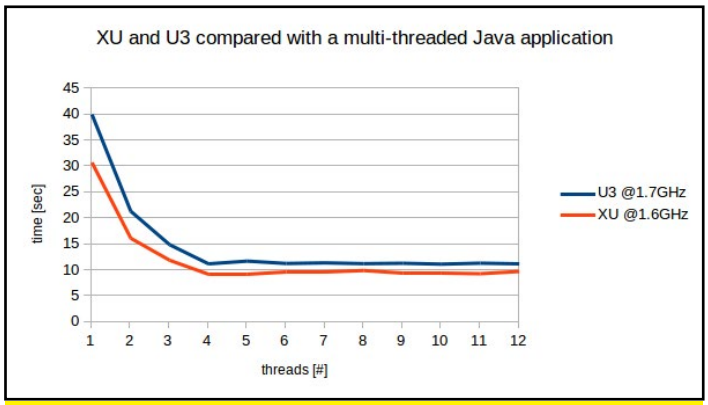


Figure 1 - XU and U3 compared with a multi-threaded Java application

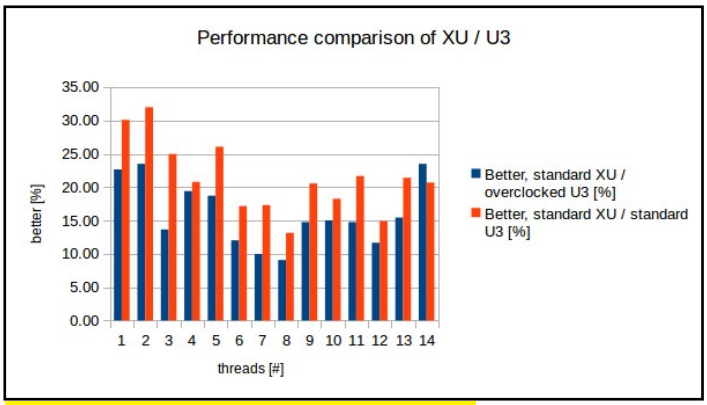


Figure 2 - Performance comparison of XU/U3

be easily overclocked by adding the following line to the /etc/rc.local file:

```
echo 1920000 > /sys/devices/system/cpu/cpu0/cpufreq/scaling_max_freq
```

We also performed a similar test run with 1.92 GHz overclock and show graphically how much faster or better XU is when compared to U3, as seen in Figure 2.

Now we can conclude that with the U3 overclocked , the XU is 15 – 20 % faster. However, we are not yet done. If we repeat the performance test and draft a figure of many trials with the overclocked U3, we get the flattened curve shown in Figure 3.

Performance decreases when tests are repeated back-to-back (without a cooldown period) at an overclocked frequency. The performance is steady with three first cores loaded, but then decreases when the 4th core is also fully utilized. The XU instead gives similar performance in all repeated tests at its default 1.6 GHz frequency. Therefore, we should also consider the thermal behavior and frequency scaling of each platform. The used frequency can be easily inspected manually with the command:

```
cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_cur_freq
```

The temperature of cores in XU are stored in a file in Linux and it can be shown as follows:

```
cat /sys/devices/platform/exynos5-tmu/temp
```

The U3 temperature can be found by checking a similar file:

```
cat /sys/class/thermal/thermal_zone0/temp
```

The value 50000 means temperature of 50 degrees. We implemented reading of temperature and current clock frequency into our test application, and therefore we were able to also collect the related thermal and clock frequency data right after the execution of each sub run at different thread configurations. Figure 4 shows the superimposed temperature and clock frequencies.

When the temperature of the chip increases, the clock frequency decreases, but increases again towards maximum frequency, when the temperature gets lower. Therefore, the U3 board keeps the temperature at a level of about 80C degrees. This behavior is very consistent, and avoids overheating very effectively and keeps the board stabilized. Therefore, the performance governor is the appropriate default setting for man-

Figure 3 - Repeated U3 test runs at I.92 GHz overclocked frequency

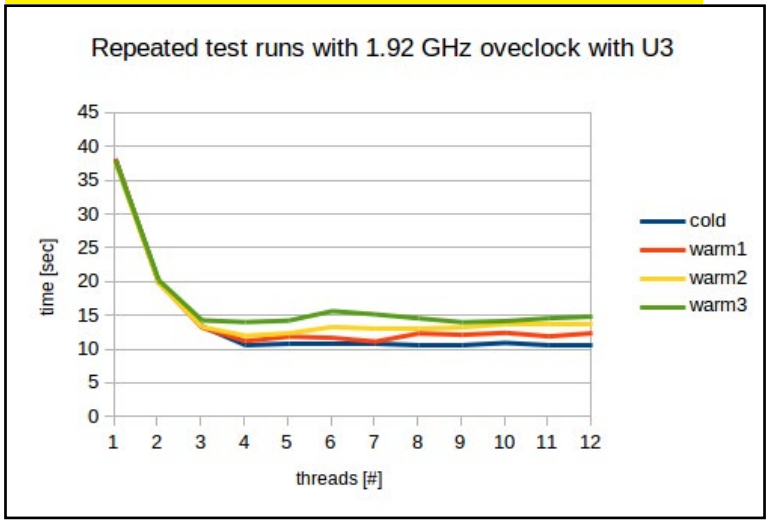
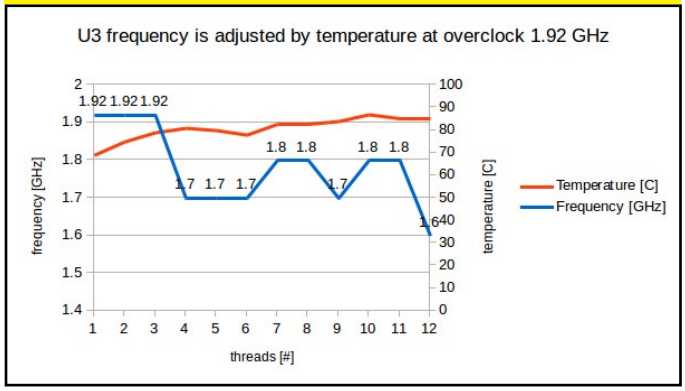


Figure 4 - U3 frequency adjusted by temperature at I.92GHz overclocked frequency



aging the clock frequency of the U3, and we did not need to take the time to try the "ondemand" governor with the U3.

To make a fair comparison between the A9 and A15 processors, both boards are run at 1.7 GHz. The XU shows better performance with all thread configurations from one thread to 12. The temperature of U3 gets hotter over the whole test run, from 60 degrees up to 78 degrees. However, the temperature of XU increases faster than the U3, from 62 degrees up to 88 degrees, which shows that the XU runs hotter than the U3 at the same frequency. On the other hand, the temperature of the XU decreases more quickly after the test is over. This is due to the included fan, which keeps rotating until the temperature gets below 55 degrees. Since a stock U3 has just a heat sink and no fan, its temperature remains higher for longer time after the test run has ended.

The XU can be configured to use only LITTLE cores by using two different means: 1) by configuring cluster switching, or 2) by setting maximum scaling frequency to no more than 600 MHz [ME13]. In both of these cases, the "ondemand" frequency governor must be used.

The first method of using the LITTLE cores is done issuing the following commands as root:

```
  echo ondemand > /sys/devices/system/cpu/cpu0/cpufreq/
scaling_governor
  # only LITTLE
  echo 01 > /dev/b.L_operator
  # it is better to wait a while and inspect that the only
A7 cores are in use
  cat /dev/bL_status
  # the output will be as follows
        0 1 2 3 L2 CCI
  [A15]  0 0 0 0  0
  [A7]   0 1 0 0  1
  # to disable cluster switching
  echo 00 > /dev/b.L_operator
```

When we want again enable also A15 cores and cluster switching we can type the following:

```
echo 11 > /dev/b.L_operator
```

The second method of isolating the A7 cluster is to set the processor frequency as root:

```
  echo ondemand > /sys/devices/system/cpu/cpu0/cpufreq/
scaling_governor
  echo 600000 > /sys/devices/system/cpu/cpu0/cpufreq/scal-
ing_max_freq
```

The board treats low frequency values as doubled, so the effectively used value is 1.2 GHz, although 600 MHz is shown. See, for instance, the output of the following cat command:

```
cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_cur_freq
```

The raw value 600000 means that current frequency is 2 x 600 MHz = 1.2 GHz.

We made a test comparison of A7 and A15 clusters by running both at 1.2 GHz to see the difference in performance and temperature purely based on architecture. The left illustration in the above figure shows the performance and temperature of LITTLE cluster. Performance improves as expected when more threads are added to do the same work. At the same time, temperature remains stable at about 52-54C degrees, and the fan does not start rotating during the test.

With the "ondemand" governor, the frequency is stable at its initial maximum of 1.2 GHz. The behavior of the A15 cluster is shown in the right side of Figure 5, and its performance is about 40-50% better than the A7 cluster. The temperature rises in the beginning of the test until 4 threads or more have been added and all 4 cores have been loaded. At that point, the fan starts rotating and temperature remains stable at about 63C degrees. The clock frequency stays at 1.2 GHz both in "ondemand" and "performance" governors.

The XU board is completely silent when only the A7 cluster is being used and is still very quiet when using the A15 cores, because the fan rotates slowly. Obviously, the XU can be overclocked with higher frequencies such as 1.4, 1.6 and up to 1.7 GHz. The next figure shows comparison of tests made at 1.4 and 1.7 GHz.

The left side of Figure 6 shows temperature and performance graphs using 1.4 GHz, and the right side shows the same tests at 1.7 GHz. At 1.4 GHz, the temperature stays below 70 C and frequency is steadily at 1.4 GHz. The performance of the 1.7GHz graph shows that it is faster, but the temperature rises more. During the test with 4 threads, the fan starts to rotate faster while the temperature rises, therefore the temperature drops during the run with 5 threads.

We prepared two different graphs to find out what should we think about "ondemand" governor against a consistently high frequency on XU. In the first graph, the relation of temperature and performance is shown, and the second

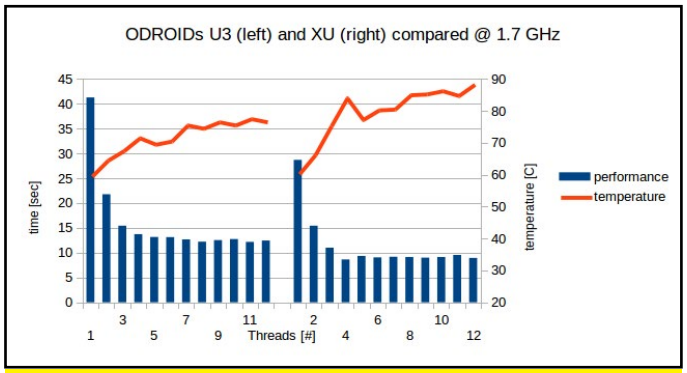


Figure 5 - XU big.LITTLE clusters @ I.2GHz, 4xA7 (left), 4xAI5 (right)

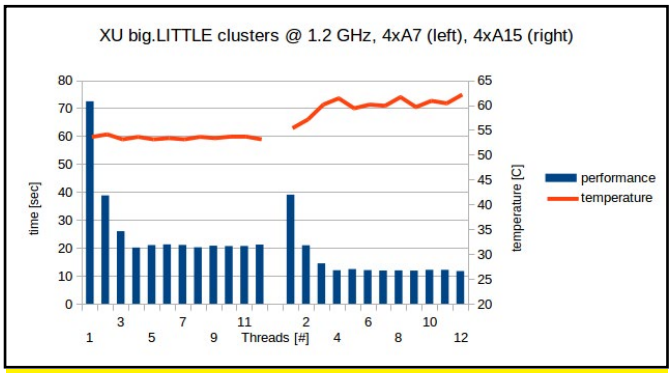


Figure 6 - AI5 @ I.4GHz (left) and @ I.7 GHz (right)

graph shows the frequency and temperature together. When a constant 1.7 GHz frequency is used, the temperature rises continuously until it is 90 degrees during the test with 12 threads. Performance is stable when all 4 cores are utilized with tests that invoke 4-12 threads. When the "ondemand" governor is used with a maximum frequency of 1.8 GHz, the temperature rises more slowly. However, performance has some degradation during tests using 6 and 9 threads.

Figure 8 shows the relation between temperature and used frequency with a test using the "ondemand" governor vs. using a constant 1.7 GHz frequency.

In the leftmost graph, the "ondemand" governor keeps temperature lower by variating frequency. The highest used frequency is 1.8 which has been read from file right after the sub run with one thread. After that, different frequencies have been used; 1.6, 1.3 and also 1.2 GHz. The right graph shows that frequency is at constant 1.7 GHz. The temperature rises more over the entire test run. Additionally, we know that the LITTLE A7 cores are not used when performance governor is on or when a constant high frequency (> 1.2 GHz) has been assigned as maximum scaling frequency. The recommendation, based on these results, is to use the ODROID-XU's "ondemand" governor for the optimum setting. Constant frequency can be assigned, or the "performance" governor used when one needs it, for instance, to study the behavior of an application under development.

Conclusions

Both computers tolerated all various test configurations very well. With the ODROID, it is safe to do different kind of overclocking experiments, because the SoCs have thermal protection against overheating.

- The processor of the XU is hotter than the the processor of the U3 at identical frequencies.
- On the basis of what has been tested here, we can conclude that the XU is 25-30 % faster than the U3. However, file IO and GPU performance have not been considered here.
- If one wants to have a perfectly silent computer, then the choice is a stock U3 without a fan.
- If one needs more power, the XU is the option to take. To address any fan noise concerns, the XU can be configured also so that fan is always off, or rotating at a low speed.
- With the XU, one gets two different computers in one single box.
- With better cooling, even higher overclocked frequencies can be attained, achieving even better performance.

The frequency behavior of U3 with the "performance" governor is easy to understand and is flawless. The frequency scaling behavior of the XU is more difficult to understand, and has not been fully covered in this article. For instance, we don't describe when and why a core is switched on or off during execution.

Figure 7 - ODROID-XU temperature vs. performance using the "ondemand" governor

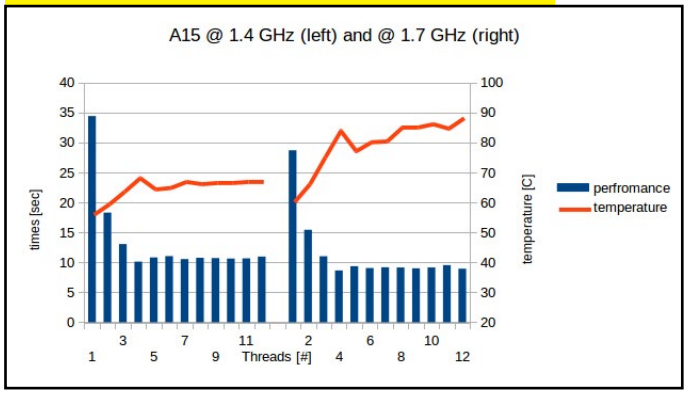
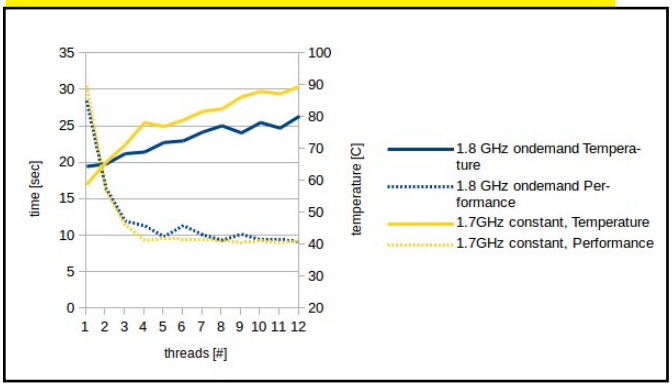


Figure 8 - ODROID-XU temperature vs. frequency using the "ondemand" governor (left) and constant frequency (right)



An impression is that there is some development potential. Let's say that XU can fully load all 4 cores at 1.6 GHz, then 100 MHz higher frequency could be used at each core count decrement. Therefore, a turbo boost governor could run at 1.9 with one core, at 1.8 with two cores, at 1.7 with three cores and at 1.6 GHz with 4 cores fully utilized. We shall see what capabilities can be leveraged in the future.

References

[ME13] Memeka. Get to Know and Control big.LITTLE, ODROID Forum, 2013. `http://bit.ly/1oOODGP`

[HK14] Hardkernel product pages, 2013. `http://bit.ly/1hD2dIn`.

[OP14] Opas J. Estimating Radio Network Interference with Multithreaded Java. ODROID Magazine, Issue #2, 2014.

SAY GOODBYE TO NANO AND SET YOUR EDITOR TO VI

by Bruno Doiche

Even now and then, you end up needing to edit your sudoers filer, and the default way to do this is issuing the visudo command and going to Vim to edit a file. So why not default your text editor once and for all?
Just use the update-alternatives command by typing this:

```
update-alternatives --config editor
```

Now, you can feel like a true hacker while learning to use Vim to edit your files. Start by using the cheat sheet that we gave you in Issue 2 on Page 27.

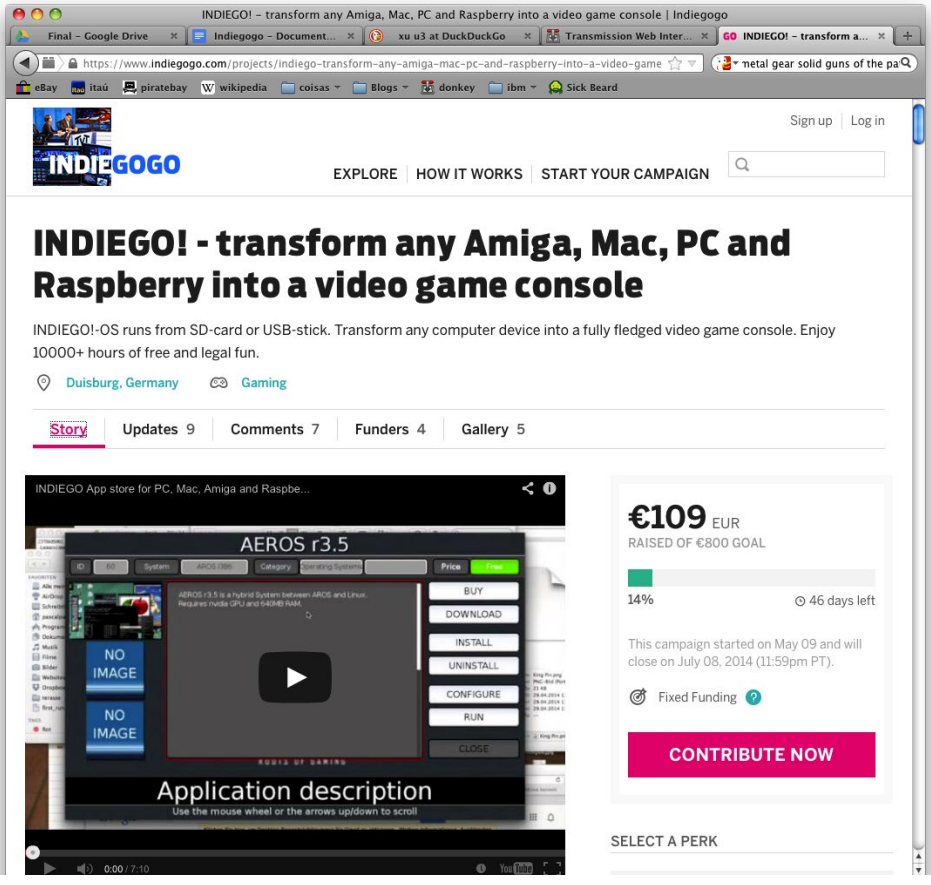
INDIEGOGO CAMPAIGN PROMISES ODROID COMPATIBILITY WITH STRETCH GOAL

INTER-INTEGRATED CIRCUITS FOR THE REST OF US

by Bo Lechnowsky

Pascal Papara, developer of the Aeros operating system, has launched a new Indiegogo campaign that ends on July 8th, 2014. It has a relatively low €800 goal ($1100 USD), with the ODROID version being developed and released at €1500 ($2000 USD). The campaign promises to convert any number of devices, including the ODROID-U3, into a low-cost system similar to Steam Box.

Supporters who donate €20 ($27 USD) will get an ODROID-U3 compatible distribution on microSD card if the ODROID stretch goal is hit. You can read more about the campaign and donate at `http://bit.ly/1nppPXT`.



Stretch goal II at 1500€:

Make a SteamOS-like Distribution for Odroid U3

includes AEROS for Odroid as well

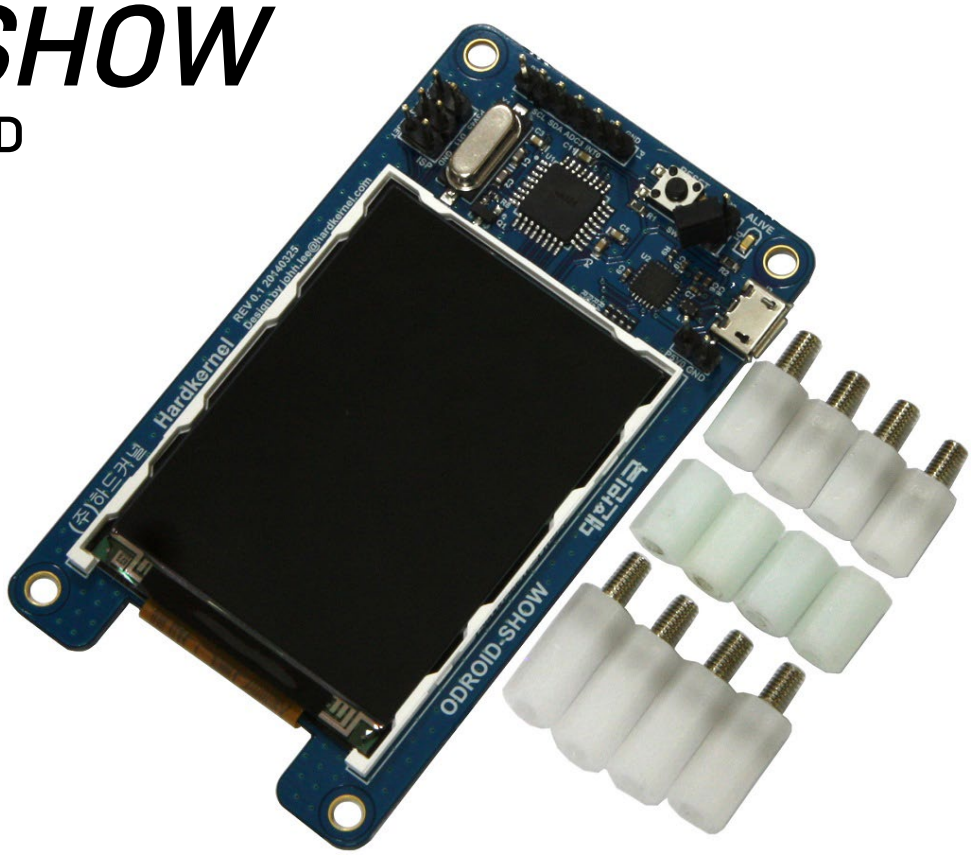
Make a SteamOS-like Distribution for SAM440/460

includes AEROS for SAM as well (SAM version will include lx command even if the last stretch goal is not reached)

ODROID-SHOW

A POWERFUL MINI LCD SCREEN FOR THE U3

By Justin Lee and John Lee



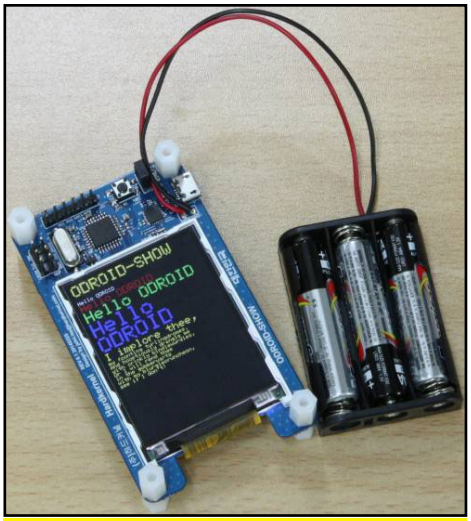
The ODROID-SHOW is a new Arduino compatible device from Hardkernel that lets you see what your ODROID or PC is thinking by using a small 2.2" TFT LCD. It is priced affordably at US$25, and is designed for stacking on the ODROID-U3. A set of PCB spacers and a and a USB cable are included.

You can show colorful texts and pictures via the USB interface with VT100/ANSI-style commands, eliminating the need to use an HDMI monitor. You can not only connect this tiny display to your ODROID, but to a Mac, Linux PC, Windows PC, or even an enterprise server as well.

The ODROID-SHOW comes with I2C, ADC, and GPIO pins for further expansion, with plans to introduce an add-on board with some specialized sensor chips for robotics applications. It can be turned into a completely portable device by adding 3 or 4 alkaline batteries. Because of very low power consumption, it is perfect for wearable projects.

ODROID-SHOW Specifications

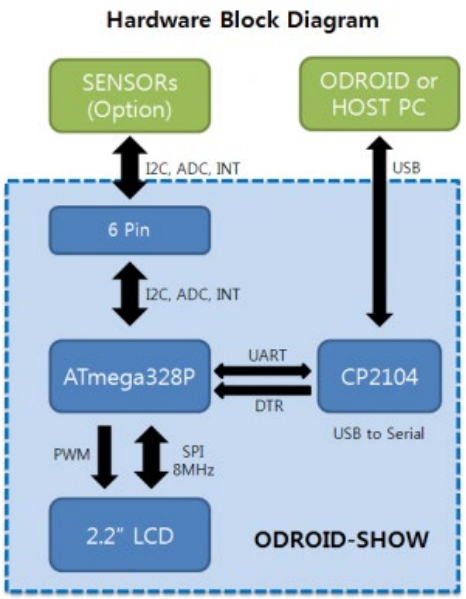
MCU	ATmega328P at 16Mhz
LCD	2.2" 240x320 TFT-LCD (SPI 8Mhz interface)
Host interface	USB to UART via on-board CP2104
Input Voltage	3.7 ~ 5.5 Volt
Power consumption	60mA @ 5Volt
Serial Port Settings	Baud rate : 500,000 bps (0.5Mbps)
	Stop bits: 8-N-1
	No H/W, S/W Flow Control
MCU/LCD Voltage	3.45 V from CP2104 on-chip voltage regulator



ODROID-SHOW with batteries

Hardware Architecture

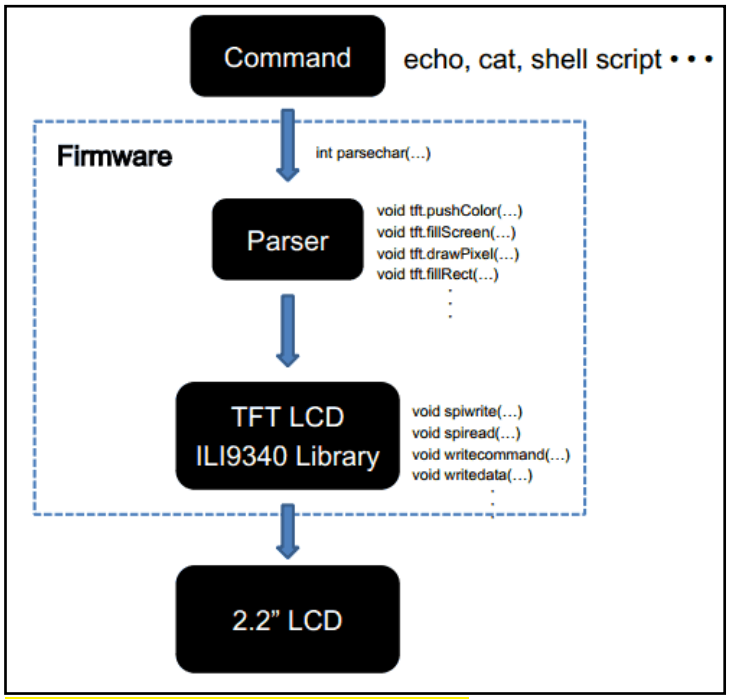
The ATmega328 is the "main brain" of the ODROID-SHOW, which can parse the stream from UART and display the data on the TFT-LCD. The UART is connected to the host PC or ODROID via CP2104, which then converts the UART to a USB interface. The CP2104 also has a 3.45V voltage regulator to supply the power for the LCD. The on-chip regulator allows for a simpler board design.



Hardware Block Diagram

The MCU contains the boot loader (Optiboot) in its flash memory and is fully compatible with the Arduino IDE. The flash memory permits changing and improving the firmware very easily.

Firmware Architecture



Firmware Architecture

The default firmware in the ODROID-SHOW was composed with the ANSI/VT100 command parser and the TFT-LCD library, and the full source code firmware can be downloaded from our GitHub repository at `http://www.github.com/hardkernel/ODROID-SHOW`. The code for the TFT-LCD library was originally developed by Adafruit, and we improved it for our purposes.

How to use it

To send information (such as text strings) to ODROID-SHOW, you need to know the ANSI/VT100 Escape Commands.

ANSI Escape Commands

Terminal codes are needed to send specific commands to your ODROID-SHOW. This can be related to switching colors or positioning the cursor.

Name	decimal	octal	hex	Description
ESC	27	033	0x1B	Escape character
CR	13	015	0x0D	Carriage return
LF	10	012	0x0A	Linefeed (newline)

Foreground coloring

ANSI	Description
Esc [3 0 m	Set foreground to color #0 - black
Esc [3 1 m	Set foreground to color #1 - red
Esc [3 2 m	Set foreground to color #2 - green
Esc [3 3 m	Set foreground to color #3 - yellow
Esc [3 4 m	Set foreground to color #4 - blue
Esc [3 5 m	Set foreground to color #5 - magenta
Esc [3 6 m	Set foreground to color #6 - cyan
Esc [3 7 m	Set foreground to color #7 - white
Esc [3 9 m	Set default color as fg color - black

Background coloring

ANSI	Description
Esc [4 0 m	Set background to color #0 - black
Esc [4 1 m	Set background to color #1 - red
Esc [4 2 m	Set background to color #2 - green
Esc [4 3 m	Set background to color #3 - yellow
Esc [4 4 m	Set background to color #4 - blue
Esc [4 5 m	Set background to color #5 - magenta
Esc [4 6 m	Set background to color #6 - cyan
Esc [4 7 m	Set background to color #7 - white
Esc [4 9 m	Set default color as bg color - black

VT100 Escape Commands

(Pn = Numeric Parameter)

VT100	Description
Linefeed(\n)	Cursor Down
Esc D	Cursor Down
Esc E	Cursor Down to row 1
Esc M	Cursor Up
Esc c	Resets LCD
Esc [Pn A	Keyboard UP Arrow
Esc [Pn B	Keyboard Down Arrow
Esc [Pn C	Keyboard Right Arrow
Esc [Pn D	Keyboard Left Arrow
Esc [Pn ; Pn H	Cursor Position
Esc [H	Cursor to Home
Esc [2 J	Erase entire screen
Esc [6 n	Reports cursor position(serial port)

Extended VT100 Escape Commands for ODROID-SHOW

Extended VT100	Description
Esc [s	Save cursor pos
Esc [u	Restore cursor pos
Esc [Pn s	Set text size (width = textsize*6, height = textsize*8)
Esc [r	Set rotation 0 to 3 (rotate to 90° in a clockwise)
Esc [0 q	Turn off LED backlight
Esc [1 q	Turn on LED backlight
Esc [Pn;Pn , Pn;Pn i	Start image-drawing mode

First, create and run the daemon service described at the end of this article before moving on to the tutorials.

Tutorial #1: Text output

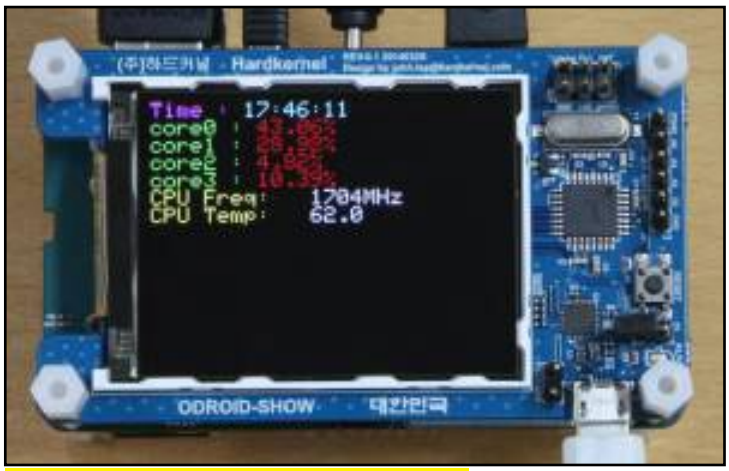
The bash script shown below can display 2 text strings with different colors and font sizes. To test, open the "/dev/ttyUSB0" port and send VT100/ANSI commands with a couple of strings:

```
#!/bin/bash

flag=0
trap "flag=1" SIGINT SIGKILL SIGTERM
./port_open &
subppid=$!
```

ODROID SHOW with text display

```
  serialPort="/dev/ttyUSB0"

  DATA[0]="ODROID"
  DATA[1]="SHOW"

  echo -ne "\e[5s\e[0r\ec" > $serialPort

  sleep 0.1

  while true
  do
    if [ $flag -ne 0 ] ; then
      kill $subppid
      exit
    fi
    for ((j=1; j<8; j++)); do
      echo -ne "\e[25;100H" > $serialPort
      for ((i=0; i<6; i++)); do
                    echo   -ne   "\e[3"$j"m\
e[3"$j"m${DATA[0]:$i:1}" > $serialPort
        sleep 0.02
      done
       echo -ne "\eE\e[55;150H" > $serial-
Port
      for ((i=0; i<4; i++)); do
                    echo   -ne   "\e[3"$j"m\
e[3"$j"m${DATA[1]:$i:1}" > $serialPort
        sleep 0.02
      done
    done
  done
```

Tutorial #2: Show your ODROID's Stats



ODROID-SHOW with statistics

This bash script shows useful ODROID statistics such as the 4-core load status, CPU frequency and CPU temperature, along with a real-time clock. To run this script, you first need to install sysstat using sudo apt-get install sysstat.

```
  !/bin/bash

  flag=0

  trap "flag=1" SIGINT SIGKILL SIGTERM

  ./port_open &
  subppid=$!

  function cpu_state {
  cpuFreqM=$(echo "scale=0; " `cat \
  /sys/devices/system/cpu/cpu0/cpufreq/
scaling_cur_freq` "/1000" | bc)
  cpuTempM=$(echo "scale=1; " `cat \
  /sys/class/thermal/thermal_zone0/temp`
"/1000" | bc)
  }

  echo -ne "\e[2s\e[3r\ec" > /dev/ttyUSB0
  sleep 0.1

  while true
  do
```

```
    if [ $flag -ne 0 ] ; then
      kill $subppid
      exit
    fi
    echo -ne "\e[0;0H\e[35mTime : \e[36m" >
/dev/ttyUSB0
    date +"%T" > /dev/ttyUSB0
    sleep 0.1
    echo -ne "\eE\eM\e[32mcore0 : \e[31m" >
/dev/ttyUSB0
    sleep 0.1
    mpstat -P 0 | grep -A1 "usr" | grep -v
"usr" | awk '{print ""$4"%   "}' > \
      /dev/ttyUSB0
    sleep 0.1
    echo -ne "\eE\eM\e[32mcore1 : \e[31m" >
/dev/ttyUSB0
    sleep 0.1
    mpstat -P 1 | grep -A1 "usr" | grep -v
"usr" | awk '{print ""$4"%   "}' > \
      /dev/ttyUSB0
    sleep 0.1
    echo -ne "\eE\eM\e[32mcore2 : \e[31m" >
/dev/ttyUSB0
    sleep 0.1
    mpstat -P 2 | grep -A1 "usr" | grep -v
"usr" | awk '{print ""$4"%   "}' > \
      /dev/ttyUSB0
    sleep 0.1
    echo -ne "\eE\eM\e[32mcore3 : \e[31m" >
/dev/ttyUSB0
    sleep 0.1
    mpstat -P 3 | grep -A1 "usr" | grep -v
"usr" | awk '{print ""$4"%   "}' > \
      /dev/ttyUSB0
    sleep 0.1
    cpu_state
    echo -ne "\eE\eM" > /dev/ttyUSB0
    sleep 0.1
        echo   -ne   "\e[33mCPU   Freq:
\e[37m"$cpuFreqM"MHz   \eE" > /dev/ttyUSB0
   echo -ne "\e[33mCPU Temp: \e[37m$cpuTempM\e
" > /dev/ttyUSB0
    sleep 1
  done
```

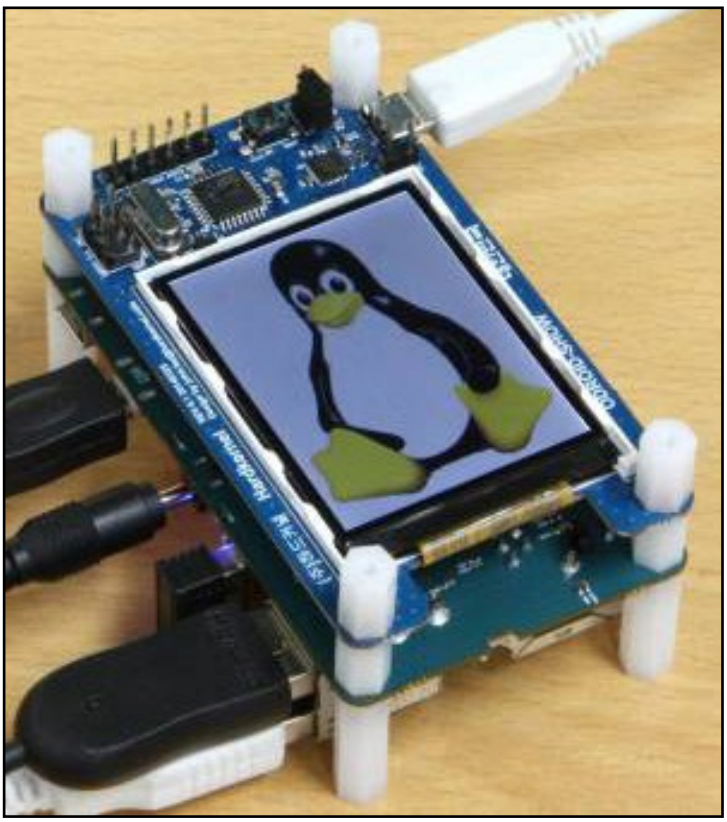
Tutorial #3: Image display

In addition to text, you can also display a graphical image on the ODROID-SHOW. In order to do so, we recommend using ffmpeg to convert a normal PNG file into a raw RGB-565 file (RGB-565 is the preferred format for compatibility). For best results, the PNG file should first be resized to fit the display, which is 240x320 pixels.

```
  ffmpeg -vcodec png -i penguin.png \
  -vcodec rawvideo -f rawvideo -pix_fmt
rgb565 penguin.raw
```

The resulting penguin.raw file will be ready for display on the ODROID-SHOW. The image load mode may be set using the pixel coordinate parameters.

```
  #!/bin/bash

  flag=0
  serial="/dev/ttyUSB0"

  trap "flag=1" SIGINT SIGKILL SIGTERM

  ./port_open &
  subppid=$!

  echo -ne "\e[0r\ec" > $serial

  while true
  do
```



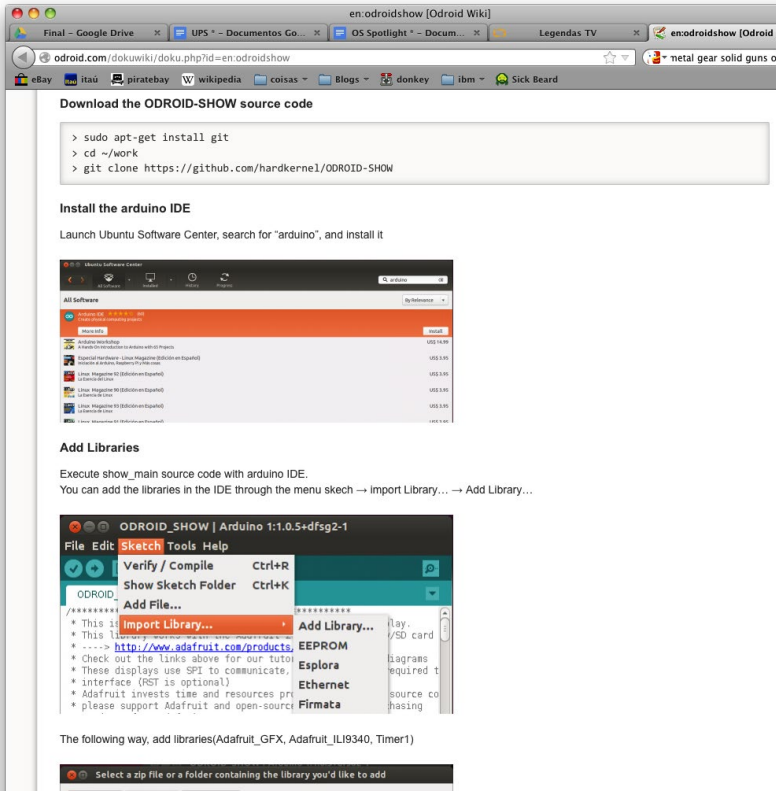
```
    if [ $flag -ne 0 ] ; then
      kill $subppid
      exit
    fi
    echo -ne "\e[0r" > $serial
    sleep 0.2
    echo -ne "\e[0;0,240;320i" > $serial
    cat penguin.raw > $serial
    sleep 0.2
    echo -ne "\e[1r" > $serial
    sleep 0.2
    echo -ne "\e[0;0,320;240i" > $serial
    cat butterfly.raw > $serial
    sleep 0.2
    echo -ne "\e[0r" > $serial
    sleep 0.2
    echo -ne "\e[0;0,240;320i" > $serial
    cat woof.raw > $serial
    sleep 0.2
  done
```

Because the "cat" and "echo" commands with redirection to the "/dev/ttyUSB0" always open and close the serial port automatically, the data flowing to the ODROID-SHOW can become corrupted. To prevent this problem, we wrote a small program which acts like a daemon to handle communication with the serial port.

```
  #include <stdio.h>
  #include <fcntl.h>
  #include <termios.h>
  #include <errno.h>

  #define baudrate    B500000

  const  char  serialPort[]  =  "/dev/tty-
USB0";

  int main(void)
  {
    int usbdev;
    struct termios options;

    usbdev = open(serialPort, O_RDWR | O_
NOCTTY | O_NDELAY);

    if (usbdev == -1)
        perror("open_port  :  Unable  to
open:");

    tcgetattr(usbdev, &options);
    cfsetispeed(&options, baudrate);
    cfsetospeed(&options, baudrate);

    options.c_cflag |= CS8;
    options.c_iflag |= IGNBRK;
    options.c_iflag &= ~( BRKINT | ICRNL |
IMAXBEL | IXON);
    options.c_oflag &= ~( OPOST | ONLCR );
    options.c_lflag &= ~( ISIG | ICANON |
IEXTEN | ECHO | ECHOE | ECHOK | ECHOCTL |
ECHOKE);
    options.c_lflag |= NOFLSH;
    options.c_cflag &= ~CRTSCTS;

    tcsetattr(usbdev, TCSANOW, &options);

    while(1)
      sleep(0.2);

    return 0;
  }
```

First, modify the serial port number in the above source code, then compile the daemon by typing gcc -o port_open port_open.c. Launch the resulting executable port_open before running any of the above example scripts to avoid data corruption during transfer to the ODROID-SHOW.

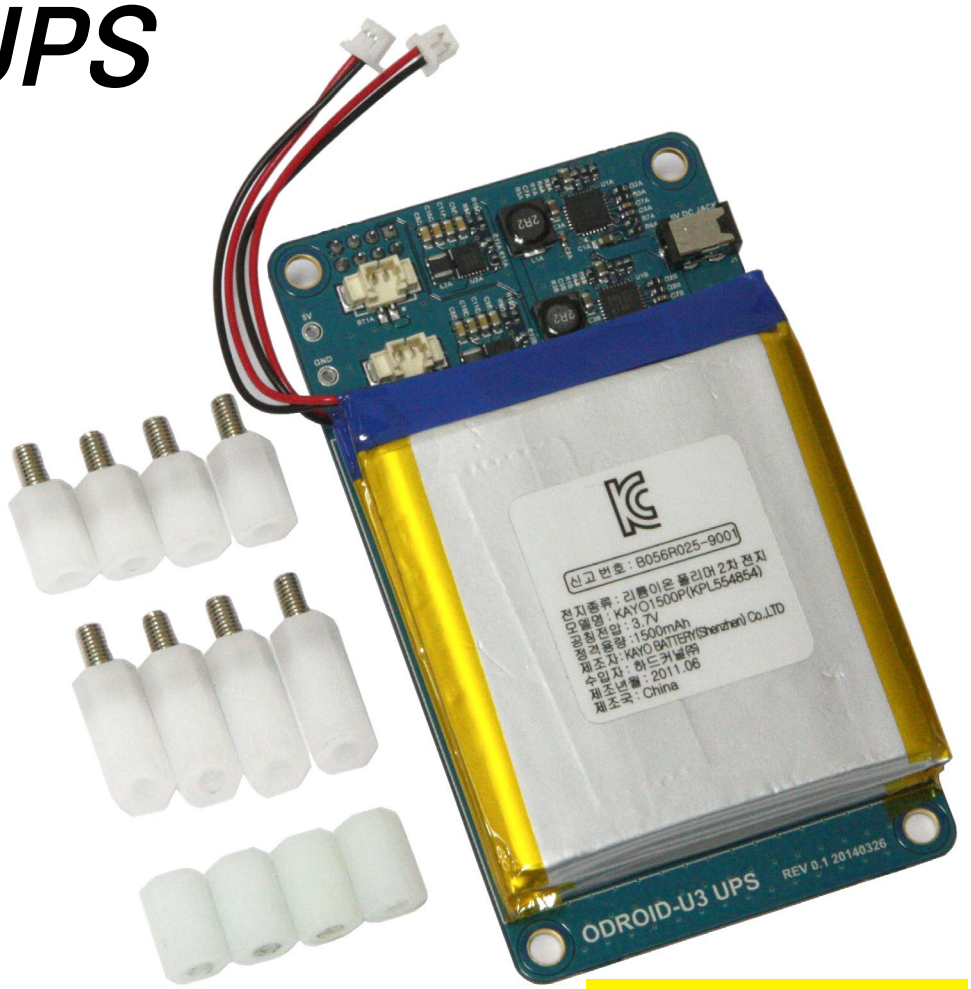
For more detailed information on setting up your ODROID-SHOW, please visit `http://odroid.com/dokuwiki/doku.php?id=en:odroidshow`.



ODROID-UPS KIT

A COMPACT AFFORDABLE BACKUP SOLUTION

by Justin Lee



This baby can extend your ODROID uptime for months!

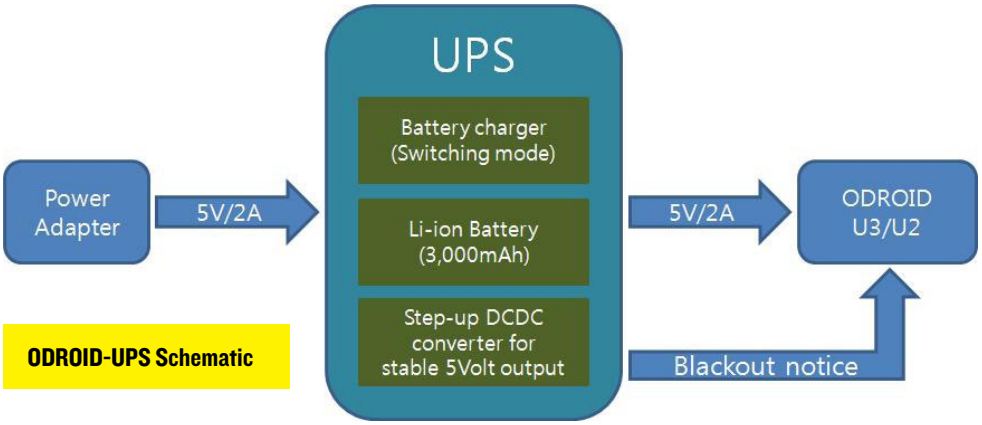
Hardkernel is proud to announce the newest addition to its family of U3 peripherals: the ODROID Uninterruptible Power Supply (ODROID-UPS). For mission-critical industrial applications, it's important to ensure that power to the ODROID remains constant in case the main power supply fails or is disconnected. Because it has the same form factor as the ODROID-U3, the ODROID-UPS stacks neatly on top of the board with PCB spacers, and connects to the 8-pin header socket on the U3.

The UPS kit contains the charger circuit, batteries and a 5V output DC-DC converter circuit. The full schematics can be downloaded from `http://bit.ly/1fDb3ds`. With a 3000mA battery capacity, the ODROID-U3 can run about 1~2 hours of heavy computing without needing AC power.

The MAX8903C is integrated using 1-cell Li+ chargers and Smart Power Selectors™ with variable power input. The switch-mode charger uses a high switching frequency to eliminate heat and protect external components. All power switches for charging and shifting the load between battery and external power are included on-chip so that no external MOSFETs, blocking diodes, or current-sense resistors are required.

The MAX8903C features optimized smart power control which makes the best use of limited USB or adapter power. The battery charger's current and SYS output current limit are independently set (up to 2A), and the system charges the battery using any leftover power from the ODROID's power adapter. Automatic input selection switches the system from battery to external power, and the DC input operates from 4.15V to 16V with up to 20V protection.

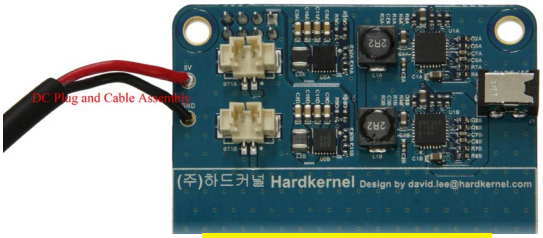
The MAX8903C internally blocks current from the battery and system back to the DC input when no input supply is present. Other features include pre-qual charging and timer, fast charge timer, overvoltage protection, charge status and fault outputs, and power-OK monitors. In addition, on-chip thermal limiting reduces battery charge rate and AC adapter current to prevent charger overheating.



ODROID-UPS Schematic

Li-ion Polymer battery

The UPS kit has two Li-ion Polymer batteries which are connected in parallel. Each battery's capacity is 1500mA for a total capacity of 3000mA. The maximum charging voltage is 4.2V.



Closeup of circuit board

DC-DC S Boost Converter IC Maxim's MAX8627

We added a boost converter because the Li-ion battery output voltage varies from 3.6 to 4.2V, but the ODROID-U3 needs a 5V input. The MAX8627 step-up converter is a high-efficiency, low-quiescent current, synchronous boost converter with True Shutdown™ and inrush current limiting. The MAX8627 generates 5V using a single-cell Li+/Li polymer battery.

If the typical voltage is 3.8V, the average battery capacity is about 11.5Wh. If the chemical and electrical efficiency runs at approximately 70%, the real capacity is 8Wh. If your system consumes 2W, the UPS kit can run for about 4 hours before needing to be recharged.

If the USB host ports of ODROID-U3 don't work, solder the DC plug cable on the UPS board and connect it to DC-Jack on ODROID-U3. It will make a stable power supply for the USB devices.

Example of automatic shutdown

The AC_OK signals are connected to GPIO199/GPIO200 in the 8-pin header socket of ODROID-U3. When a blackout or sudden disconnection of AC power happens, the system will shutdown automatically after 1 minute by using the following bash script, which continuously checks the status of the AC power.

```
#!/bin/bash
echo 199 > /sys/class/gpio/export
echo 200 > /sys/class/gpio/export
echo in > /sys/class/gpio/gpio199/direction
echo in > /sys/class/gpio/gpio200/direction

get_ac_status() {
  ac1=`cat /sys/class/gpio/gpio199/value`
  ac2=`cat /sys/class/gpio/gpio200/value`

  if [ "0$ac1" -eq 1 -o "0$ac2" -eq "1" ]; then
    export ACJACK="off"
  else
    export ACJACK="on"
  fi
}

while :
do
  get_ac_status

  if [ "$ACJACK" == "off" ]; then
    shutdown -P 1
  fi

  sleep 1;
done
```

Design note

The UPS project was created because we had an overstock of battery packs in our inventory. However, the internal protection circuit in our battery packs were too sensitive to heavy current load, so we decided to use 2 cells in parallel. Another concern was the unbalanced energy in the two separated batteries, which may lead to a chemical hazard. In order to address this, we implemented all of the circuits twice on the UPS board design, which means the schematics are not well optimized.

If you are considering making your own power bank, be sure to check the maximum output current of the battery pack first. When the electrical load is very heavy, the protection circuit in the battery pack disconnects the load automatically. To reset the protection circuit, you must unplug and replug the battery connectors. Although you can check for the maximum AC input state (AC_OK) using a script , it may not be sufficient. When using the ODROID-UPS with a power bank, you will want to include additional circuits for conveying the battery level to the intelligent power management.

For more detailed specifications and to purchase your own ODROID-UPS, please visit `http://bit.ly/1fDb3ds`.

OS SPOTLIGHT: FULLY LOADED

UBUNTU 12.11 WITH UNITY 2D DESKTOP ENVIRONMENT

by Rob Roy, Editor-In-Chief

Fully Loaded with Kernel 3.0 for the U2/U3/X/X2 may be downloaded from http://bit.ly/1rhHymu

The ODROID forum offers many excellent community prebuilt operating system images, with each one containing unique customizations for such diverse applications as media players, software development, music, and robotics. Because the ODROID family of single board computers is intended as a development platform, many users prefer to compile their own operating systems in order to have full control over all aspects of the hardware. In the previous issue of ODROID Magazine, Hardkernel developer Mauro Ribeiro presented a useful guide to building your own version of Ubuntu from source code to assist those who want to learn how to "do it yourself". But what if you just want to use your ODROID immediately, without investing the technical expertise and time in compiling your own operating system?

Fully Loaded, which was first introduced to the forums in 2013 and has been updated regularly, was one of the first community images to offer an out-of-the-box desktop experience, which eliminated the need to spend time configuring software, installing applications and debugging desktop environments. It contains nearly every flavor of Ubuntu available for 12.11, including Gnome, Lubuntu (LXDE), Kubuntu (KDE), Blackbox, Openbox, Fluxbox, Unity and Xubuntu (XFCE). You can switch the desktop environment by using the circular icon next to the username on the login screen. I personally recommend KDE Plasma Workspace for its hardware-accelerated visual effects and Windows 7-style interface, but each one has its own unique advantages.

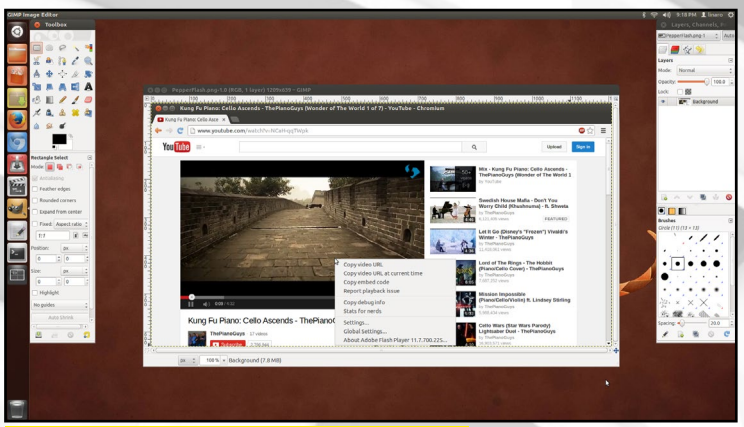
To get started, download and copy the Fully Loaded image for your hardware (X, X2, U2/U3), to an eMMC module or SD card. For more information on doing so, refer to the "Copying an Image File" article, also included in this issue of ODROID Magazine. Once the image boots, you will see the familiar Ubuntu login screen with an image of its 12.11 mascot, the Precise Pangolin. If you have your sound turned on, you will also hear the classic "Flute Flute Slap" drum sound that is unique to Ubuntu.

The default username for Fully Loaded is "linaro" with the password "linaro". After logging in, you can set up your local time zone, choose the language settings, and create users by clicking on the "System Settings" menu option in the top right corner of the desktop. The root password is also "linaro", but this login should only be used for maintenance rather than everyday use to prevent accidental damage to the file system.

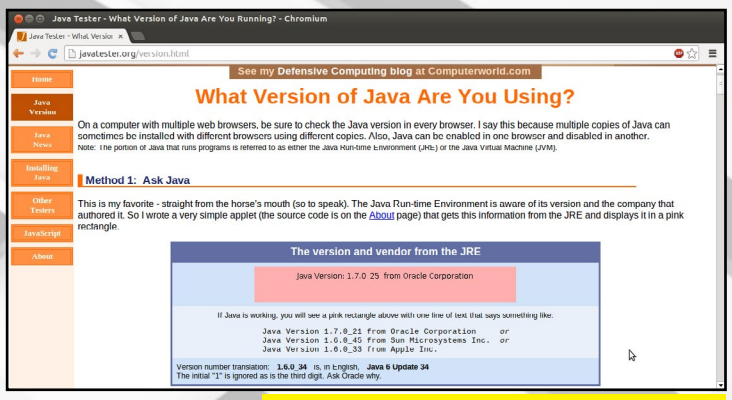
One of the first things that you'll see when using Fully Loaded is that the Unity desktop is set as the default, as it has been proven to be very popular with both beginners and experts for its friendly icons, convenient taskbar, and incomparable stability. Because the ODROID platform supports the GLES graphics library rather than OpenGL, the 2D version of Unity is preferred for its superior performance. It runs much faster than the standard 3D version by eliminating much of the extraneous "eye candy" and corresponding graphical lag that was introduced in Ubuntu 13.04.

The desktop environment is a matter of personal preference, since your choice of environment doesn't affect the software library available. Ubuntu is designed to remain consistent in its applications while allowing complete customization of the graphical user interface (GUI). This article presents is an overview of the major applications available on Fully Loaded, all of which are open-source and freely available from the Ubuntu Software Center.

Ubuntu 12.11 is one of the most stable operating systems available for the ODROID



GIMP, the GNU Image Manipulation Program, has a long development history.



Java applets in the web browser offer true cross-platform compatibility.

GIMP (GNU Image Manipulation Program)

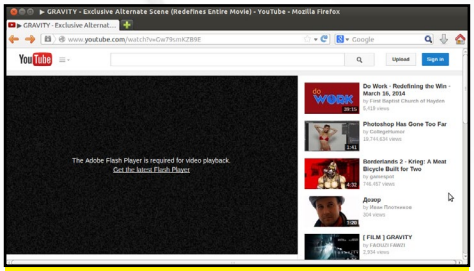
Who needs to buy Photoshop when you can have one of the most powerful graphics program available for absolutely no cost? GIMP has been in active development since 1996, and includes a rich library of user-submitted enhancements and contributions. It requires a moderate amount of expertise to use, but the results can be amazing.

GIMP has just about everything that Photoshop offers, with the ability to create, modify, and enhance and save to many image formats, including JPG, GIF, PNG, PSD, and AutoCAD. The familiar toolbox on the left side contains buttons used for area selection, brushes, text editing, color swapping, masking, cloning, and shapes. Many visual effects are also available from the central window's Filters menu, such as blurring, sharpening, noise, edge detection, shadows, and other useful graphics processors.

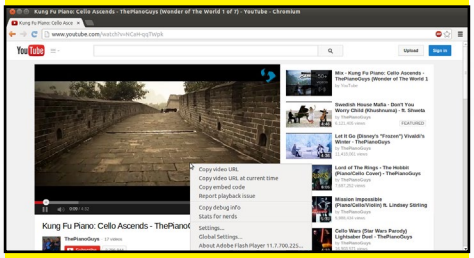
GIMP also includes a powerful plugin called "Script-Fu", based on a language called Scheme. You can design your own visual effects and processors using complex mathematical transformations, then share your work with others. More information on using Script-Fu to enhance GIMP can be found at `http://bit.ly/1fBPgTA`.

Web Browsers

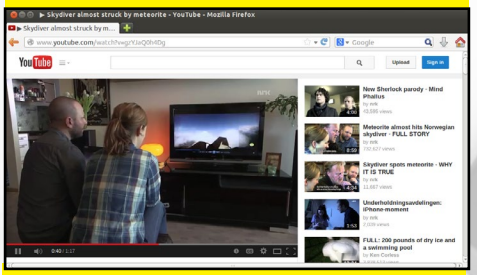
No modern computer environment would be complete without a web browser. Fully Loaded includes both Firefox and Chromium, which both give a full-featured browsing experience with support for both Java and Flash. The open-source plugin called IcedTea is enabled in both Firefox and Chromium, which gives Java applets the ability to run inside a browser. Adblock Plus is also installed in both applications, which prevents advertisements and pop-ups from interfering with your Internet session.



Oh darn! No Flash. Really?



Yaaay! Flash is here, but not from Adobe.



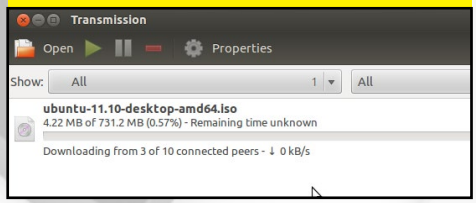
Look Ma! We can watch stuff without Flash!

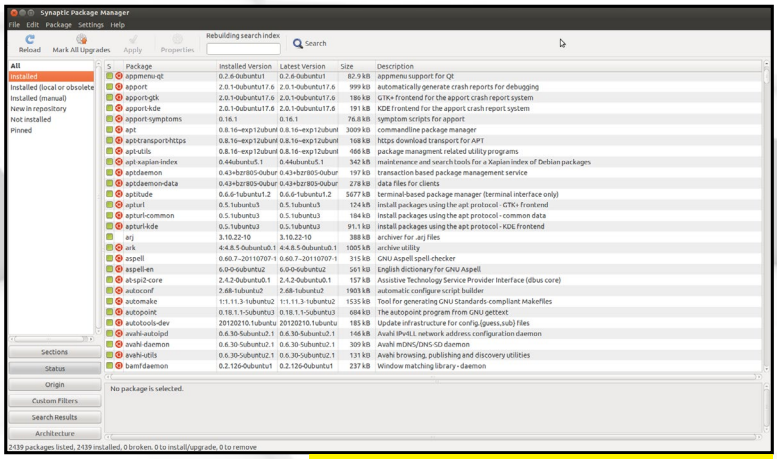
When browsing Youtube or other similar sites, both Chromium and Firefox support the recently added HTML5 player, but not all videos on the Internet are available in this updated format. Although Firefox doesn't include an open-source Flash player, the optional PepperFlash plugin for Chromium replaces the standard Flash player and allows ARM devices such as the ODROID to play Flash-based videos, even though Adobe no longer supports it.

Transmission

Transmission is the standard Linux client for the BitTorrent protocol, which allows files to be downloaded from a peer-to-peer network, yielding much higher download speeds by utilizing a network of machines rather than accessing a single computer. To use Transmission, start Firefox and navigate to any website that offers torrents, then click on the Magnet link to download the torrent file. Transmission will automatically launch and start the download, saving it to the Downloads directory once completed. Chromium can also be configured to use torrents, but Firefox is already associated with Transmission by default.

Torrent, the way to go to get all the things!





The graphic interface way to apt-get? Synaptic of course!

Xine is so cool, that we will list here all that it supports for your playback needs:
•Physical media: CDs, DVDs, Video CDs[6]
•Container formats: 3gp, AVI, ASF, FLV, Matroska, MOV (QuickTime), MP4, NUT, Ogg, OGM, RealMedia
•Audio formats: AAC, AC3, ALAC, AMR, FLAC, MP3, RealAudio, Shorten, Speex, Vorbis, WMA
•Video formats: Cinepak, DV, H.263, H.264/MPEG-4 AVC, HuffYUV, Indeo, MJPEG, MPEG-I, MPEG-2, MPEG-4 ASP, RealVideo, Sorenson, Theora, WMV (partial, including WMVI, WMV2 and WMV3; via FFmpeg)
•Video devices: V4L, DVB, PVR
•Network protocols: HTTP, TCP, UDP, RTP, SMB, MMS, PNM, RTSP
(and the crowd cheers for the largest screenshot caption ever!)

Xine and ffmpeg

If you want to watch a downloaded video on your ODROID, Xine is the best application available for 12.11, and supports many popular formats including .mp4, .avi, and .mkv (Matroska). Although it's software-decoded, most 720p videos will play very well on the U3, and 1080p videos are watchable even though some frames will be dropped. Because hardware video decoding is not available with Kernel 3.0, the updated Ubuntu 13 Dream Machine with XBMC image should be used in cases where 1080p video on Linux is desired.

To use Xine, simply double-click on any video from the File Manager, and press "g" to display the HUD which contains seek controls, volume buttons, and other useful features. It is essentially a wrapper for the powerful ffmpeg video player utility. Fully Loaded includes a special version of ffmpeg compiled specifically for the NEON architecture of the Mali GPU. FFmpeg may also be invoked without Xine by typing ffmpeg on the command line.

Synaptic Package Manager

Synaptic is the main application for downloading new software packages and upgrading existing ones. It offers thousands of development libraries, full packages, desktop environments, and much more. If you are using the ODROID for software development and wish to install any missing dependencies, this is the place to find them. There are many hidden gems available in Synaptic, if you take the time to look through the enormous list of open-source packages. The password for starting Synaptic is the root password of "linaro".

Ubuntu Software Center

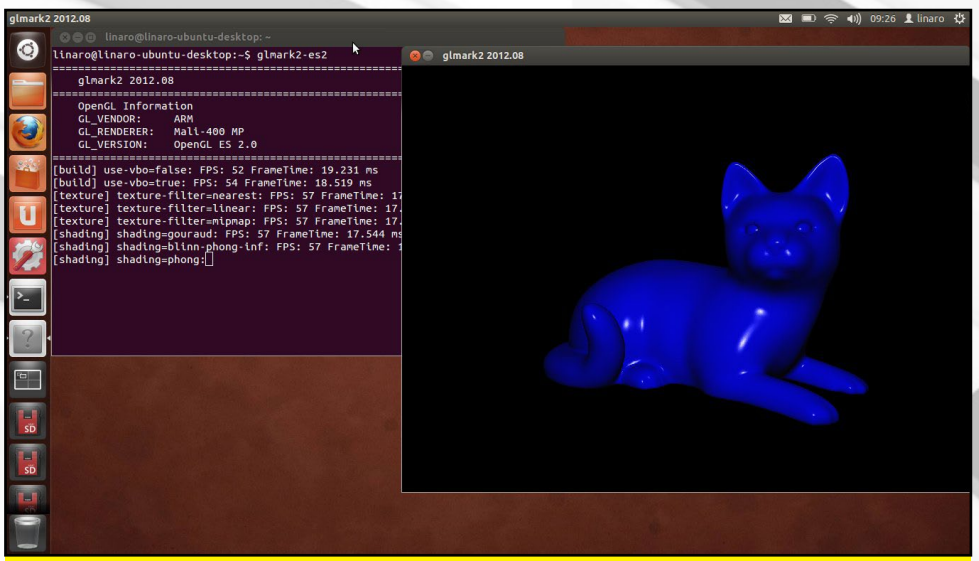
The Ubuntu Software Center is a user-friendly interface to the Canoncial software repositories, and offers similar packages to the Synaptic Package Manager, but in a friendlier format. Its advantage over Synaptic is that the software is categorized and includes short explanations of the purpose of each application, but does not include development libraries in its lists. It is the equivalent of iTunes for OSX or the Google Play Store for Android.

Terminal

Many how-to articles in both the forums and this magazine require entering strings of commands into the Terminal window, which is the standard Command Line Interface (CLI) that comes with almost all Linux distributions. Several useful commands include sudo, which permits a command to be run with superuser access, ls, which shows the contents of a directory, and cd, which navigates to a specified folder. Type cd ~ to go to your home folder, and press Tab to use the auto-complete feature. You can also press the Up arrow to reuse recently typed commands. A useful shortcut for Terminal in most Ubuntu desktop environments is pressing the key combination Ctrl-Alt-T.

Mali 3D Drivers

The Mali GPU included with the X, X2, U2 and U3 has great 3D capabilities, and you can run a graphical test by typing either es2gears or glmark2-es2 in the Terminal window. The jellyfish animation in glmark2-es2 is especially nice! Game and graphics developers should become familiar with the OpenGL ES 2.0 commands in order to program for the ODROID, which is an optimized subset of the original OpenGL language.



Is that a screenshot showing the simultaneous use of Terminal and Mali 3D driver, or a secret reference to our cat-napping article from an earlier issue?

Oracle Java Development Kit (JDK8)

Fully Loaded comes with Oracle JDK8 installed, which allows Java programs such as Minecraft Server to be run from the command line. The Java Virtual Machine can be invoked by typing java in the Terminal window. Many useful programs that have been written for other hardware will generally also run well on the ODROID, since Java is a platform-independent language.

Mednafen

There is an enormous library of games available for the ODROID, and Mednafen supports many different emulated systems, including Gameboy, NES, SNES and Sega. A convenient script comes with Fully Loaded called play_rom, which automatically sets the optimum values for scaling and resolution in Mednafen. ROM files can be either double-clicked from the File Manager, or invoked using the play_rom <rom file> command in the Terminal Window. For detailed information on Mednafen and its supported formats, refer to the How-To guide at `http://bit.ly/1pYi1hu`.

Other Tips and Tricks

To boot directly to the most recently used desktop environment, type:

```
sudo /usr/lib/lightdm/
lightdm-set-defaults
--autologin linaro
```

in the Terminal window. This command bypasses the login screen which saves time when rebooting frequently. To change the default environment, simply log out of the current desktop and choose another one at the login screen.

To get even more speed and performance from your ODROID, Fully Loaded allows overclocking to 1.92GHz by typing `sudo gedit /etc/rc.local` in the Terminal window and removing the "#" from the beginning of the line that starts with "echo 1920000". It's highly recommended to use a fan when overclocking to prevent shutdowns due to overheating.

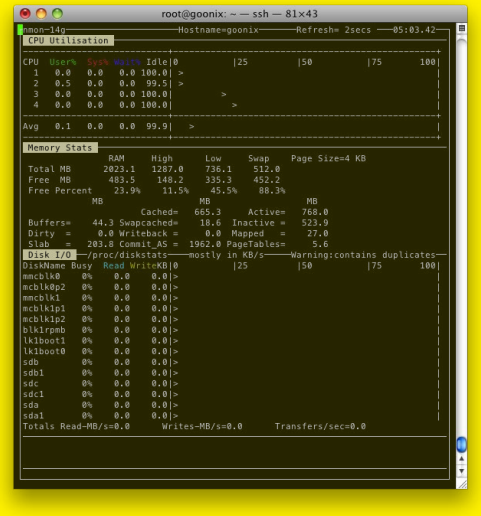
If you own more than one ODROID, a useful kernel swap script is included on the Fully Loaded image to allow booting from a single MicroSD shared between a U2 and an X2. To do so, type `./media/boot/tools/swap_odroid.sh [u2 | x2]` in the Terminal window before shutting down. For more details, please refer to `http://bit.ly/1lIDHtQ`.

The Fully Loaded kernel also includes Virtual Memory support to extend your ODROID's memory above 2GB. To learn more about the swap file and how to enable it, visit `http://bit.ly/1pYfWSY`.

MONITOR YOUR LINUX WITH NMON

by Bruno Doiche

Are you still using top to monitor your overall system statistics? Try using NMON. It is a great tool to monitor everything, from your processes to your network connections in a single handy program. Plus, it can monitor your system and export the data as a .csv file to create detailed reports of your ODROID's long-term performance!



`sudo apt-get install nmon`

To collect data, run nmon like the example below (-f means that the nmon will log a data file, -s is the time between refreshes and -c the count of refreshes nmon will do to end the data collection) that will run for 1 hour:

```
nmon -f -s 30 -c 120
```

Nmon will create a file in your current directory:

```
<hostname>_date_time.nmon
```

When run with these flags, nmon does not show the graphical interface itself, but instead runs as a background job, disconnected from your shell, so that you can logoff while it collects your data for future review.

Happy data analysis!

BUILD AN ODROID-POWERED OFF-ROAD UNMANNED GROUND VEHICLE

PART I: OVERVIEW, PLATFORM ASSEMBLY, AND POWER DISTRIBUTION

by Christopher D. McMurrough

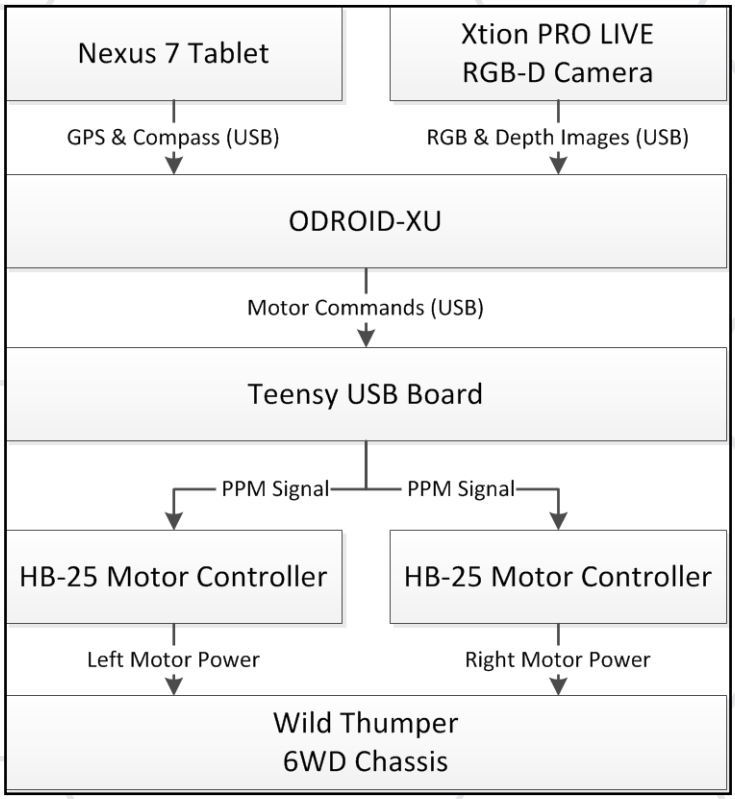
If this were a classic magazine sold at a newsstand, this picture of Chris' UGV would require us to use drool-proof paper!

In this series of articles, we will build our very own off-road Unmanned Ground Vehicle (UGV) using the ODROID-XU development board. Our goal will be to create a robot that is capable of traversing outdoor terrain while moving between GPS waypoints, and also to provide the reader with a solid platform for future development. We will use navigational data provided by an external Android device and 3D scene information provided by an RGB-D camera. The series will be divided into 3 articles covering general platform design and power distribution, interfacing motors and sensors with the ODROID, and programming the robot to autonomously follow GPS waypoints.

Introduction

I am going to assume that you, the reader, must really like ODROIDs (after all, you are reading ODROID Magazine). Chances are, you probably like robots too. The good news is that ODROIDS are perfect for robots! However, keep in mind that building your own robot is always challenging. Whether you are just getting started on your own or have worked on robots before, each robot is different and requires many different considerations for the project to be a success. This series is intended to provide an overview of creating a complete UGV system using as many commercially available, off-the-shelf (COTS) components as possible. We will cover the mechanical, electrical, and software design aspects of our system, and provide as much supplemental material as possible such as schematics and source code. In this first article, we will focus on the general mechanical assembly and power distribution of our system. While this is not intended to be a fully comprehensive "how-to" guide showing every step of the build process, I encourage questions and comments regarding this article series in the "ODROID Robot-

Component	Qty	Price	Source
ODROID-XU	1	169.00	Hardkernel
Wild Thumper 6WD Chassis	1	249.95	Pololu
Nexus 7 Tegra 3	1	149.99	Newegg
Asus XTION PRO LIVE	1	169.99	Newegg
HB-25 Motor Controller	2	49.99	Parallax
M2596 Buck DC-DC Adjustable PSU	1	3.45	Amazon



UGV System Architecture

ics" discussion in the ODROID Forum (Board index / Hardkernel / General Chat / ODROID Robotics). I will respond to questions and provide more details as requested. My intention for the series is to start an active robotics discussion within the ODROID community, so please join in!

System Overview

Our robot, when complete, will be able to move between sequential GPS waypoints while avoiding obstacles. We will use GPS and compass input from a Nexus 7 Android tablet for position sensing, and 3D information from an Xtion Pro Live RGB-D camera for obstacle avoidance. These devices both provide lots of information that we don't necessarily need for our waypoint following demonstration, but may be useful in future projects. The ODROID-XU running Ubuntu Linux will process information from these devices using Robotic Operating System (ROS), which we will discuss more thoroughly in the next article of the series.

The chassis that we will be using is the 6WD Wild Thumper. This platform is ideal for off-road environments, given that each of the 6 motor and wheel assemblies feature independent suspension. The motors are designed to work with standard 7.2 volt RC battery packs, and the chassis has room to accommodate 4 such batteries under the top mounting plate. We will power our chassis with 3 NiMh battery packs (rated at 3000 mAh) wired in parallel, giving us a total of 9000 mAh of motor power. A fourth battery pack will be dedicated to powering the ODROID, sensors, and other electronics. Separating this battery from the others provides our electronics with a layer of electrical isolation, and will prevent our system from resetting due to motor current related power fluctuations. The electronics battery will be regulated to a clean 5 volts using an LM2596 DC-DC power supply.

We will use dual HB-25 motor controllers to drive the 6 motors on the Wild Thumper chassis. The 3 left motors will be connected to one HB-25 in parallel, while the 3 right motors will be connected to the other. The stall current of each of the 6 motors is 6.6 Amps, which will require each HB-25 to provide a maximum of 19.8 Amps. The HB-25 can provide a maximum of 25 Amps, but we will replace the provided fuse with one rated for 20 Amps. This is not mandatory, but it will ensure that the HB-25 does not provide more than 20 Amps in the event of a short circuit. If at any time in the future we notice fuses blowing frequently, we will know that the motors are drawing more than the specified maximum amount of current and can troubleshoot accordingly.

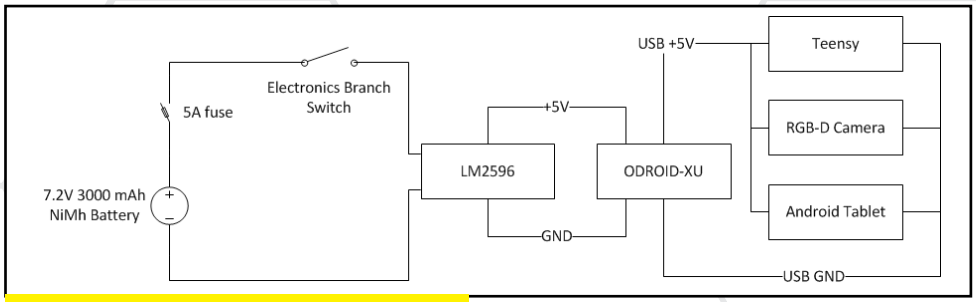
The final electrical component that we will be using is a Teensy USB Board. This microcontroller allows us to generate the control signals for the HB-25 (PPM servo pulses), as well as interface with additional sensors and components in the future. We won't do anything more than mechanically mount this device for now, but in Part 2 we will explore device interfacing in much greater detail.

Platform Assembly

The 6WD chassis, once assembled according to the manufacturer instructions, is ready to accommodate our sensors and devices. We will be mounting the RGB-D camera and motor controllers directly to the top chassis plate. The motor controllers are each placed on a pair of aluminum spacers and secured with 6-32 screws. The RGB-D camera will also be mounted with a pair of 6-32 screws by drilling two small holes through the plastic base.

The ODROID-XU, Teensy, LM2596 module, and power switches will be mounted inside of a rugged enclosure. The enclosure we will be using is a plastic equipment case, which is mounted to the vehicle chassis using 4 1-1/2" aluminum spacers and 6-32 screws. The

With its built-in ODROID-XU, this UGV is able to do its fair share of exploring while also offering amazing processing power.



Electronics power branch

height of the spacers prevent the rear wheels from rubbing against the enclosure when driving over rough terrain, since the equipment case is a bit wider than the top chassis plate (but narrower than the wheelbase). We will similarly secure our electronics to the enclosure using screws and spacers, and will drill small holes in the case as needed to create the mounting points.

Using the equipment case allows us to shield the electronics from the outdoors, but we will need to take care to ensure that the enclosure is properly cooled. For now, we will just prop the lid open until we finish the rest of the assembly and setup. Before we start doing actual field testing in Part 3, we will add cooling fans to regulate the internal temperature of the case.

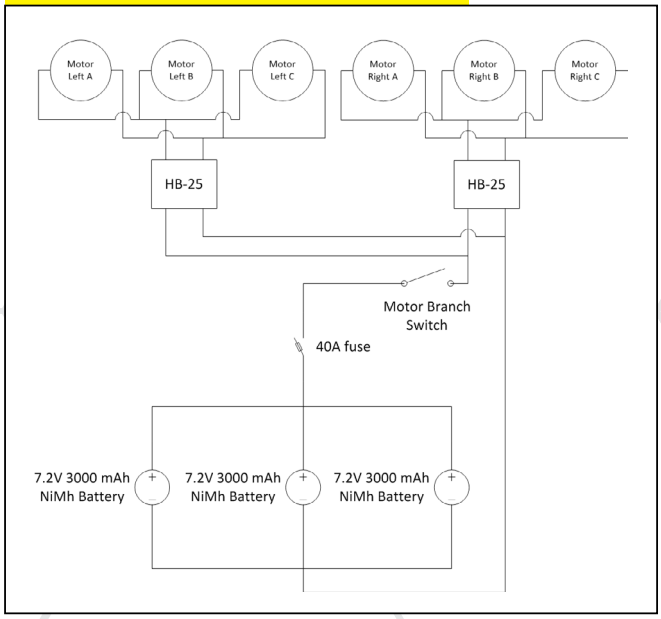
Power Distribution

As previously mentioned, we will be supplying power to the case electronics (regulated by the LM2596) and motor controllers using separate battery sources, so we will mount two switches to the enclosure. This is important for two reasons. First, we want to ensure that the Teensy is generating control signals before the motor controllers are provided power. If the controllers are powered before receiving a proper signal, the driver state cannot be guaranteed and could possibly result in the motors being provided power in an uncontrolled manner. Second, there will be times during development when we don't want to provide power to the motor side branch at all (an example being when the robot is sitting on your desk while the ODROID is connected to a monitor and keyboard). Using two switches will allow us to disable either power branch independently. We will also place inline fuse holders on each power branch to protect against short circuits. The fuse for the motor branch will be 40 Amp, while the electronics branch will be 5 Amp. The HB-25 controllers each have their own 20 Amp fuse, but we will include a master fuse on each branch as an added layer of protection.

Once the LM2596 regulator is connected to the electronics branch battery, we need to set the voltage to a steady 5V before connecting it to the ODROID-XU. Additionally, we must solder a barrel connector to the output solder pads of the device so that we can power the ODROID, and indirectly, the additional devices which are powered from the USB bus. Barrel connectors can be found at some electronics stores, but the receptacle on the ODROID-XU is fairly common and can easily be salvaged off of an old DC power supply. After soldering your cable to the LM2596, use a voltmeter to verify the polarity of the connector. Next, with the voltmeter attached to the barrel connector, adjust the potentiometer on the LM2596 until it reads a steady 5.0 volts. It is critical that the polarity and voltage level is checked before connecting the ODROID! Once you have verified the polarity and voltage level, you can connect the PSU power jack to the ODROID when you are ready to power it up.

In an earlier development prototype, the UGV used an XBOX 360 Kinect.

Motor power branch



Conclusion

This article is the first installment of the our 3 part series on ODROID powered robotics. In Part 2, we will focus on getting Linux and Robot Operating System (ROS) running on the ODROID-XU and interfacing with our devices to control motors and read sensors. Be sure to follow the forum discussion for more details on our robot project at `http://forum.odroid.com`.

MEET AN ODROIDIAN

SIMONE (@SERTOO), A LONG-TIME ODROID ENTHUSIAST AND HELPFUL COMPUTER EXPERT

edited by Rob Roy

The man himself, @sert00 is everywhere to be found at our forums.

Please tell us a little about yourself.

I'm a 27 year-old industrial electrician, working as an electro-mechanical operator in a big company in Italy alongside more than 2700 people. We process and sell food, in particular chicken meat. For the most part, my job is about finding and repairing all the problems with the machines involved in the production line, such as communication, automation, motors and pneumatic systems.

How did you get started with computers?

I started with computers at the age of 9. My father would bring me the PCs, printers, monitors and peripherals that were obsolete at his work and destined for the trash bin. I began with DOS, then used Windows Workgroup 3.11.

What types of projects have you done with your ODROIDs?

I initially bought my first ODROID, a U2, without any use of it in mind. I love the ARM world, followed Android development for some years, and then decided to try an Android-based board to see how it worked. In the meantime, I also switched to using Linux after years of using Windows, and thought it would be a good start with the U2, since my old laptop broke.

I then won another U2 from Hardkernel as a monthly award, which was mostly used for learning how to do things like recompile the kernel for Android and my phones and tablet. I also went deeper with Linux by using it as a web and media server.

Last year, I received from Hardkernel an XU-E beta version (rev. 0.2) as an engineering sample, then bought another XU-E (rev. 0.3), which more reliably handled input voltage spikes. With it, I tried everything possible, from creating a media center to tinkering with the electronics. What I like about ODROIDs is the fact that I can try something, use it, and then change the configuration and focus on something else.

The last board that I bought was the U3, which is awesome. Sadly, I haven't had enough free time to use it. However, my goal will be to do what I did with my old Arduino by controlling some things in my bedroom like the lights and TV using a Java APK over a wireless connection. I put the project on hold because I needed the Arduino for other things.

I have all the addons and gadgets offered by Hardkernel, and especially like the touchscreen that I use with my XU-E running Android from 64GB eMMC. It's sort of a homemade tablet. In fact, that will be my next project when I have some time available, borrowing some ideas from Mauro's article (in the April issue) where he presented a guide for building a rugged tablet.

A small sample of @sert00's workbench. He also has a gigantic monitor on the wall and lots of network routers.

What other hobbies and interests do you have?

I like to walk in nature and ride my mountain bike in the spring season. Where I live, there are many great places for that.

What do you like most about the Odroid community?

I really like the kindness of the Hardkernel staff like Justin, Lisa and Mauro. All of the forum members are there to help out, and there are some very talented and skilled people in the ODROID community.

Are you involved with any other software or hardware projects?

Recently, I've been focusing on something different that I always wanted to learn. I bought an S7-200, analog module and ethernet module from Siemens, and I'm studying PLC programming. PLC will be useful in my work, and I also plan to use it to automate my house in conjunction with my ODROID-U3 and IO shield.