

ODROID

Magazine

Year One
Issue #8
Aug 2014



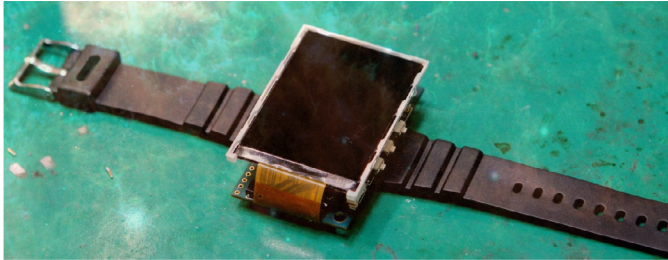
INTRODUCING THE ODROID-W

**HARDKERNEL'S MICRO-SIZED PI-COMPATIBLE
BATTERY-POWERED WEARABLE COMPUTER!**

OS SPOTLIGHT:

**POCKET ROCKET AND
COUCH POTATO**

**THE ULTIMATE ANDROID
GAMING IMAGE WITH
1080P VIDEO PLAYBACK**



**MAKE YOUR
OWN ODROID
SMARTWATCH**

- THE NEXT-GENERATION ODROID-U3+
- STEP-BY-STEP BASICS OF LINUX KERNEL COMPILATION
- INSTALL A WEB SERVER WITH NGINX AND LIGHTTPD

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



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





Hardkernel had a busy month, and released several new products that once again prove that they are the innovative leaders in ARM micro-computing. Not only has Justin and his team developed a new model for both the U and XU series, but they have also created a tiny wearable Raspberry Pi-compatible product that packs extra features.

The ODRROID-VU is the USB touch-screen that you've been waiting for!

A 9-inch model with 10-point multi-touch input will be available in the middle of August, and will not only work with the ODRROID family of computers, but is also compatible with any Windows, Ubuntu and Android machine that supports USB monitors.

The ODRROID-W is a tiny version of the Raspberry Pi that is intended for embedded, battery-powered applications such as wearables and micro-controllers. It offers many new features not included with the original Raspberry Pi, including removable USB ports, while remaining very affordable.

And finally, the long-anticipated ODRROID-XU3 is now available, and it is the fastest ODRROID ever made! It's got everything the first-generation XU-E offers, with some extra features:

- Samsung Exynos5422 Cortex™-A15 2.0Ghz quad core and Cortex™-A7 quad core CPUs
- Mali-T628 MP6 GPU supporting OpenGL ES 3.0/2.0/1.1 and OpenCL 1.1 Full profile
- Built-in energy sensors like the XU-E

We'll be reporting more about the ODRROID-XU3 in the October issue, but you can get your XU3 now at the Hardkernel Store at <http://bit.ly/lprRcPL>.

In Linux kernel news, forum user @dsd gives us a report on the newest 3.17 kernel, which appears to now have ODRROID support! Follow the original thread at <http://bit.ly/ls50v7z>:

“Samsung has been doing a load of work upstreaming ODRROID support into official Linux. The first patches will go into Linux-3.17, although a load more patches are needed to make this usable as a desktop system - might take a few more releases before all patches are polished and included.

They are also working on upstreaming ODRROID support into uboot, and they have a nice feature there, they have found a way to automatically detect X2 vs U2/U3 (by checking for the extra LED on the X2). So now uboot will automatically pick the right device tree to load. That means one appropriately crafted “OS image” will work for X2, U2 and U3.”

This issue has lots of information for hardware enthusiasts, including an in-depth look at the different types of eMMC modules that Hardkernel makes and a hands-on look at the powerful ODRROID-SHOW. We also feature a guide to compiling your own kernel directly from the Hardkernel GitHub repository, a tutorial for installing the light-weight Nginx web server on an ODRROID, Nanik begins his fascinating series on Android Application Development, and more!

ODRROID Magazine, published monthly at <http://magazine.odroid.com/>, is your source for all things ODRROIDian. Hard Kernel, Ltd. • 704 Anyang K-Center, Gwanyang, Dongan, Anyang, Gyeonggi, South Korea, 431-815 Makers of the ODRROID family of quad-core development boards and the world's first ARM big.LITTLE architecture based single board computer. Join the ODRROID 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'm a computer programmer living and working in San Francisco, CA, designing and building web applications for local clients on my network cluster of ODROIDS. My primary languages are jQuery, Angular JS and HTML5/CSS3. I also develop pre-built operating systems, custom kernels and optimized applications for the ODROID platform based on Hardkernel's official releases, for which I have won several Monthly Forum Awards. I use my ODROIDS for a variety of purposes, including media center, web server, application development, workstation, and gaming console. You can check out my 100GB collection of ODROID software, prebuilt kernels and OS images at <http://bit.ly/1fsaxQs>.



**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.



**Manuel
Adamuz,
Spanish
Editor**

I am 31 years old and live in Seville, Spain, and was born in Granada. I have recently become a father, and my son is now 5 months old. It is an incredible experience! A few years ago I worked as a computer technician and programmer, but my current job is related to quality management and information technology: ISO 9001, ISO 27001, and ISO 20000. I am passionate about computer science, especially micro computers such as the ODROID, Raspberry Pi, etc. My other great hobby is mountain biking, and I occasionally participate in semi-professional competitions.



**Nicole Scott,
Art Editor**

I am currently a Digital Strategist and Transmedia

Producer who specializes in online optimization and inbound marketing strategies, social media directing and team coordination, as well as media production for print, TV, film, and web. I also have experience in graphic and website design, social networking management and advertising, video editing and DVD authoring. I own an ODROID-U3 which I use to run a sandbox web server, live in the California Bay Area, and enjoy hiking, camping and playing music. Check out my web page at <http://www.nicolecscott.com>.



**Bruno Doiche,
Art Editor**

Is enjoying his time off on vacation for the month of August.



INDEX



ANDROID DEVELOPMENT - 6



MOUNT YOUR INTERNAL SD CARD - 9



INTRODUCING THE ODROID-W - 10



SEARCH WITH GOOGLE BBS / FIXING ANDROID OVERSCAN - 14



ALL ABOUT HARDKERNEL'S EMMC MODULES - 15



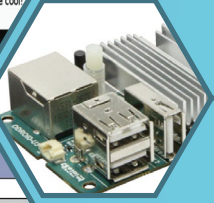
LINUX KERNEL COMPILATION - 17



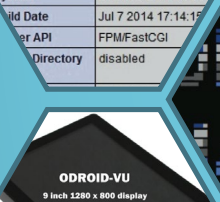
YOUTUBE PLAYER ALTERNATIVE - 21



INTERESTING LINUX COMMANDS PART 1 - 22



ODROID U3 VS ODROID U3+ - 23



INSTALL A HOME WEB SERVER - 25



INTERESTING LINUX COMMANDS PART 2 - 25



ODROID-VU AFFORDABLE 9" USB HDMI TOUCH SCREEN - 30



PEPPERFLASH CHROME PLUGIN FOR LUBUNTU 14.04 - 32



ANDROID GAMING: MUPEN64PLUS - 33

IO SHIELD DEMYSTIFIED - 34

DIGGING (INTO) THE ODROID-SHOW - 36

OS SPOTLIGHT: POCKET ROCKET AND COUCH POTATO - 41

MEET AN ODROIDIAN: BO LECHNOWSKY - 47

ANDROID DEVELOPMENT: USING THE LINUX KERNEL

A GUIDE TO THE ANDROID-SPECIFIC DRIVERS

by Nanik Tolaram

Linux is the heart of Android, and it depends heavily on it for its operation, and without it Android would not be able to run. However, Android does not use a “plain vanilla” Linux kernel, which means you cannot download a kernel from www.kernel.org and immediately use it to run Android. There are a number of drivers, code and configuration options to be added in order for Android to work properly, and this article will walk through those specific drivers and configuration, as well as discuss the drivers that are needed to ‘Android’-ify the Linux kernel upon which the Android framework depends. Most of the driver code that needs to be added is located inside the `kernel/drivers/staging/android` folder as can be seen in Figure 1.

There are also a number of drivers that are located outside the base directory `/staging/android` that are already in the Linux Kernel mainline. This article is by no means an exhaustive list for all of the drivers needed, but is a good starting point to get a better understanding of the different drivers needed for Android.

Below are the configuration options that need to be included as part of the kernel zImage. The following configuration is taken from the file `odroidu_android_defconfig` inside the `kernel/arch/arm/configs` folder, as shown in the following table.

```
CONFIG_ANDROID=y
CONFIG_ANDROID_BINDER_IPC=y
CONFIG_ANDROID_LOGGER=y
CONFIG_ANDROID_RAM_CONSOLE=y
CONFIG_ANDROID_RAM_CONSOLE_ENABLE_VERBOSE=y
CONFIG_ANDROID_RAM_CONSOLE_ERROR_CORRECTION=y
CONFIG_ANDROID_RAM_CONSOLE_ERROR_CORRECTION_DATA_SIZE=128
CONFIG_ANDROID_RAM_CONSOLE_ERROR_CORRECTION_ECC_SIZE=16
CONFIG_ANDROID_RAM_CONSOLE_ERROR_CORRECTION_SYMBOL_SIZE=8
CONFIG_ANDROID_RAM_CONSOLE_ERROR_CORRECTION_POLYNOMIAL=0x11d
CONFIG_ANDROID_TIMED_OUTPUT=y
CONFIG_ANDROID_TIMED_GPIO=y
CONFIG_ANDROID_LOW_MEMORY_KILLER=y
CONFIG_ASHMEM=y
```

If you look inside the Makefile, which is located inside the `drivers/staging/android` directory, you will see the kernel drivers that are specific to Android. In the following sections we are going to take a closer look at the different drivers inside this directory, starting with the binder driver.
















 binder.c	101.9 kB	C source code
 binder.h	9.1 kB	C header
 Kconfig	2.4 kB	plain text document
 logger.c	15.1 kB	C source code
 logger.h	1.7 kB	C header
 lowmemorykiller.c	5.7 kB	C source code
 lowmemorykiller.c.1	5.6 kB	C source code
 Makefile	301 bytes	Makefile
 ram_console.c	12.4 kB	C source code
 timed_gpio.c	4.4 kB	C source code
 timed_gpio.h	815 bytes	C header
 timed_output.c	3.1 kB	C source code
 timed_output.h	1.1 kB	C header

Figure 1 – Driver list inside the `/staging/android` directory

binder.c

The binder driver is the main backbone driver for applications in Android, and is the gatekeeper of communication between different processes that are running, written in native (C/C++) or Java.

Without this driver, Android applications will not work at all, since it mounts the essential `/dev/binder` virtual filesystem. Android uses the binder driver extensively, and because applications utilize this framework, any application is indirectly reliant on the binder driver.

```
obj-$(CONFIG_ANDROID_BINDER_IPC) += binder.o
obj-$(CONFIG_ANDROID_LOGGER) += logger.o
obj-$(CONFIG_ANDROID_RAM_CONSOLE) += ram_console.o
obj-$(CONFIG_ANDROID_TIMED_OUTPUT) += timed_output.o
obj-$(CONFIG_ANDROID_TIMED_GPIO) += timed_gpio.o
obj-$(CONFIG_ANDROID_LOW_MEMORY_KILLER) += lowmemorykiller.o
```

Example odroidu_android_defconfig configuration

ashmem.c

Location: /kernel/mm/

The name ashmem stands for Android Shared Memory, and as the name implies, is a driver that facilitates memory sharing between processes, and provides a mechanism for Linux to reclaim memory if it finds itself under memory pressure. Ashmem makes the /dev/ashmem directory available for applications to access its functionality.

The driver works using a file descriptor which can be shared by processes by utilizing the binder driver that was discussed previously. The unique characteristic of ashmem is that when the process dies, the memory goes with it, so there will not be any orphaned memory. It is also smart enough to allow applications to specify which memory area can be reclaimed or not, which is known as unpinning and pinning.

In order to save memory space, the unpinned pages are periodically reclaimed by using the LRU algorithm. Ashmem has its own shrinker that is called when available memory gets low. The normal use case on using ashmem is shown in the float chart in Figure 3, to the bottom-right.

wakelock.c

Location: /kernel/power

This is the a controversial driver that started the heated discussion between the Android kernel team and Linux kernel developers back in the days when Google had just starting committing its code for the mainline kernel, and was considered by purists to be a hack. This driver gives userspace applications the ability to re-

quest certain services that use always-on hardware so that they still operate when the device goes to sleep in order to conserve battery power.

The wakelock driver is built on top of Linux power management driver. Linux provides power management functions such as suspend and resume, where it allows the device to be 'suspended' by running on a low power mode, or 'resumed' by coming back from sleep. The wakelock driver is a communication mechanism allowing user space applications and system tasks to determine whether the device is allowed to go to suspend mode. The driver will not suspend the device if there are active wakelocks in the system.

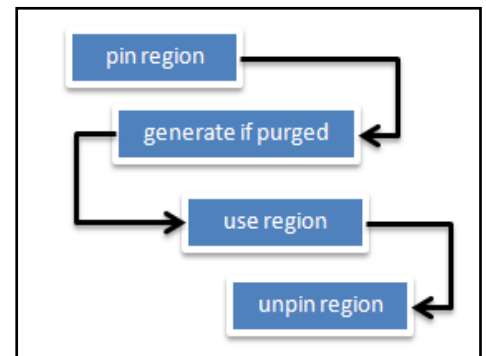
Imagine if you are running a music player on your mobile phone and the system decided that it's time to suspend, your music player stopped playing. Or, imagine if the system is about to suspend and there is an incoming phone call, but you can't receive it since the system is in the process of suspending. Wakelock makes it possible to avoid the above described scenarios, allowing you to use your device without having to worry when it will go to suspend mode. The wakelock driver introduced a new virtual filesystem called /proc/wakelocks.

Power management, wakelock and

early suspend drivers work together as one unit to manage power resources in an Android device. The sample screenshot shows the content of the wakelock virtual filesystem inside the Nexus 7.

ram_console.c

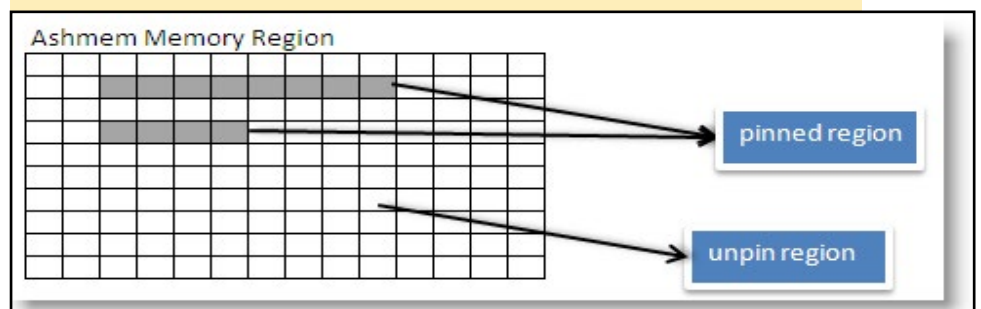
The RAM console drive can be labelled as a Linux log helper driver. Its function is to save the content of the



Ashmem pinning and unpinning flowchart

Linux log to a different memory location before the device reboots. The content of the Linux log (which normally is accessed using dmesg) can be viewed after reboot in the /proc/last_kmsg virtual file system. If the device shuts down completely with no power supplied to it, the last Linux log will not be stored the next time you startup the device. This is because the device is completely shut down and there is no power to retain the content of the memory. As long as there is power supplied to the device, the next time you restart, you will be able to extract the last Linux log through the last_kmsg virtual filesystem. This driver is very

Figure 3 - Ashmem regions



```
root@android:/sys/module/wakelock/parameters # cat /proc/wakelocks
name count expire_count wake_count active since total time sleep time max time last_change
power-supply" 35 0 0 690183515 665284482 22190375 1989782564706
main" 7 0 0 179313595648 0 95068307404 1983210289001
PowerManagerService" 62 0 0 0 45461460309 18975918653 25024409255 1983120599001
wlan_wake" 1115 0 0 3374948560 323453592 1088035000 1983044848001
alarm" 39 0 0 1158069467 1111682461 44861324 1973444387002
KeyEvents" 124 0 0 559400187 98981467 335318000 1973261028002
event2-358" 16 0 0 3676100 2735098 785188 1973260746002
event0-358" 35 0 0 9258000 0 6291000 1813541323002
alarm_rtc" 1 0 0 16929000 0 16929000 21404972002
event1-358" 0 0 0 0 0 0 0
bluesleep" 0 0 0 0 0 0 0
elan_touch" 0 0 0 0 0 0 0
power-supply" 2 0 0 83000 0 48000 2491732002
low_battery_detection" 0 0 0 0 0 0 0
power-supply" 1 0 0 125000 0 125000 1994648003
charger_configuration" 1 0 0 0 27480000 0 27480000 1990818003
wake_lock_dockin" 0 0 0 0 0 0 0
suspend_backoff" 0 0 0 0 0 0 0
unknown_wakeups" 0 0 0 0 0 0 0
deleted_wake_locks" 0 0 0 0 0 0 0
usb_config_wake_lock" 1 0 0 1987773182880 1987773182880 1815341403234 1987773182880 7215835002
cable_state_changed" 1 1 0 0 4996373567 0 4996373567 2491682002
mmc0_detect" 1 1 0 597118566 0 597118566 3430937003
mmc1_detect" 2 1 0 903140566 0 599122567 5028933002
wlan_ctrl_wake" 1 1 0 3327369567 0 3327369567 15840686002
wlan_rx_wake" 25 25 0 39758636715 13886512978 5562600001 1820382314002
```

The wakelocks drivers area considered a Google hack by purists

useful for troubleshooting issues such as kernel panic, out of memory or any other kind of problem that causes a kernel panic.

earlysuspend.c

Location: /kernel/power

You can see this driver in action when you leave your device inactive for few minutes and the screen turns off by itself. It's responsible for making sure all the necessary components inside your device are 'suspended' to conserve energy. This driver and wakelock go hand-in-hand. It relies on Linux's power management driver, and can be triggered by sending state information to the following /sys/power/state filesystem.

If you have root access to your device, you can trigger the early suspend by executing the following command:

```
echo mem > /sys/power/state
```

logger.c

The logger driver is the central logging driver used by Android system, which is also made available to user applications via the built-in logger API. Internally, the driver splits the logging

into different categories: main, events, radio and system. The following virtual filesystem are made available from the driver:

```
/dev/log/system
/dev/log/radio
/dev/log/events
/dev/log/main
```

Android provides a log viewer application called logcat that allows us to view the log in a more human readable format. The screenshot shows a sample of the content of the /dev/log/radio in its raw form, which is not very easy to read

lowmemorykiller.c

This driver takes care of killing processes from memory when the amount of free memory hits a certain threshold. Every time an application is launched, it is assigned a value to indicate what kind of application it is. This 'marker' will indicate to the driver whether this process can be killed.

As an example, looks at the phone application that you use to receive and make calls. The application will be 'marked' to a value that indicates to the driver to kill the application only when the last low memory threshold is

hit, which means that only when the memory falls to the lowest free memory available than the application will be killed. The following table lists the different value that the framework will use to assign to launched process:

- HIDDEN_APP_MAX_ADJ
- HIDDEN_APP_MIN_ADJ
- SERVICE_B_ADJ
- PREVIOUS_APP_ADJ
- HOME_APP_ADJ
- SERVICE_ADJ
- BACKUP_APP_ADJ
- HEAVY_WEIGHT_APP_ADJ
- PERCEPTIBLE_APP_ADJ
- VISIBLE_APP_ADJ
- FOREGROUND_APP_ADJ
- PERSISTENT_PROC_ADJ
- SYSTEM_ADJ
- MIN_HIDDEN_APPS

Internally, the driver only accommodates a maximum of 6 different values, so the framework does an internal conversion to fit it to the available slots. The different values can be found in the /sys/module/lowmemorykiller/parameters/adj virtual filesystem.

The settings of free available memory threshold are linked in a 1:1 ratio with the adj value, and it is stored in the /sys/module/lowmemorykiller/parameters/minfree virtual filesystem.

The minfree value is in pages of 4k. For example, if it says the free memory is 8192, it means (8192 * 4k) = 32768 which converts to 32KB (32768 / 1024).

The way in which the minfree threshold is mapped with the value from adj is explained in this chart:

Minfree threshold mapped with adj value	
adj	minfree
0	8192
1	10240
2	12288
4	14336
9	16384
15	20480

INTRODUCING THE ODROID-W

A MINIATURE RASPBERRY PI-COMPATIBLE WEARABLE COMPUTER

by Justin Lee

The ODROID-W is a miniature wearable computing module which is fully compatible with all software available for the Raspberry Pi (RPI).

The **W** stands for:

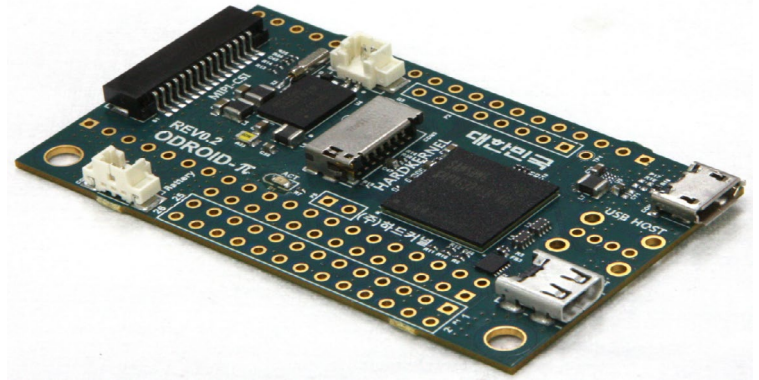
Wearable device development

Widely applicable Internet of Things (IoT) development

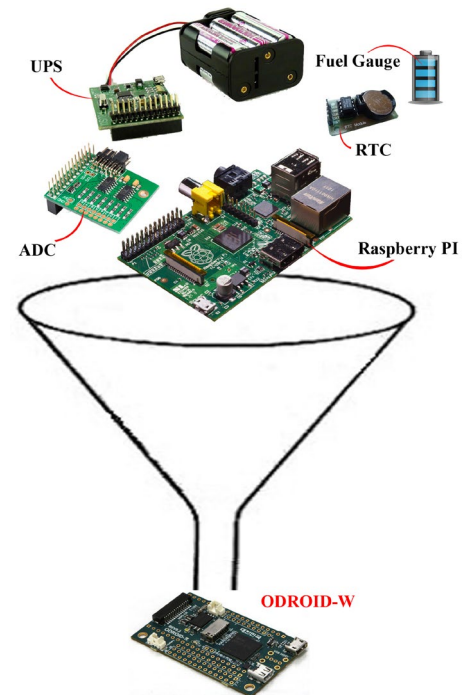
Workable DIY electronics prototyping

The ODROID-W measures a very small 60 x 36 x 7mm (2.4 x 1.4 x 0.3"). It also includes many new features and improvements over the original Pi:

- Li-polymer rechargeable battery charger and fuel gauge circuit for wearable and robotics applications
- Real Time Clock (RTC) to keep accurate time without an Internet connection by adding a coin battery
- 12-bit precision ADC to measure the dynamic voltage signals via two single-ended inputs
- DC/DC step-down converters for higher power efficiency



An early revision of the ODROID-W, which was nicknamed the ODROID-Pi



The ODROID-W adds extra features to a fully compatible Raspberry Pi clone

- DC/DC step-up converter for 5Volt rails (USB host and HDMI) from a Li-Polymer battery
- USB Host port can be placed on top or bottom as preferred
- DIY friendly 100mil/2.54mm pitch GPIO ports (up to 32 ports) for handy prototyping

The ODROID-W = RPI + RTC + ADC + UPS + Battery Gauge with significant minimalism.

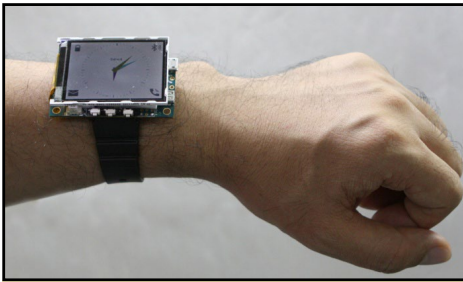
Development History

In early 2014, we had an important project with our partner companies to help them with prototyping a few wearable & Internet of Things (IoT) devices.

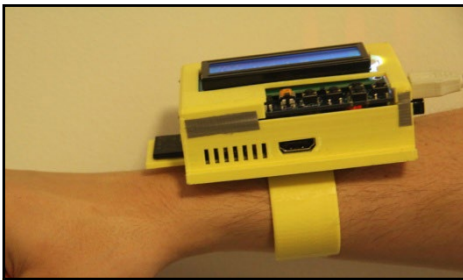
We first considered using the ODROID-U3 as a base platform. Although the ODROID-U3 is 8-12 times faster than the Raspberry Pi, the power consumption of the U3 isn't suitable for wearable devices like watches or necklaces. We even considered using the Raspberry Pi itself due to the lower power requirements and nice Linux BSP support, but the PCB of the RPi was huge (much bigger than the ODROID-U3).

To create the smallest wearable ac-

cessory possible, we decided to make our own (tiny) version of a Raspberry Pi, which allows full use of many widely available Pi peripherals such as the Pi Camera module connector and 26-pin GPIO port. The HDMI port and SD slot were changed to micro-sized connectors, and the MIPI DSI port was removed due to limited PCB space. In addition to its smaller size, the ODROID-W maintains



This prototype of a U3 smartwatch required too much power.



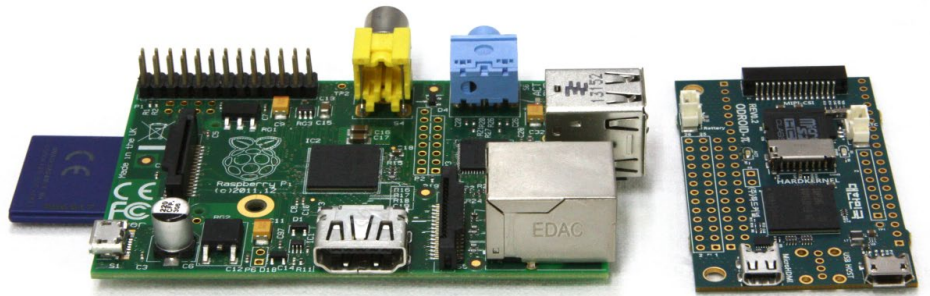
The Raspberry Pi smartwatch was far too big to be viable.

full compatibility with all existing Raspberry Pi software and peripherals.

The first sample ODROID-W PCB was designed on April 14, 2014, and had a few jumper wires. The second run was designed on May 19, 2014, which corrected some of the electronics designs and added an eMMC module socket for test purposes, even though the eMMC is not much faster than SD card due to the slow eMMC host speed in the SoC. In testing the eMMC speeds, we discovered only ~10% improvement. The third sample (rev 0.3) was thoroughly tested, and is ready for mass production.

The unit price for the ODROID-W is US \$30 and it will be available for purchase from the Hardkernel Store at <http://www.hardkernel.com>.

We didn't follow the instructions carefully enough and got this instead.



Side-by-side size difference comparison between the Raspberry Pi and the ODROID-W without the USB attachment

Processor: Broadcom BCM2835 ARM11 700Mhz

Memory: Samsung 4Gbit (512MB) LPDDR2 SDRAM

PMIC: Ricoh RC5T619 includes DCDCs, LDOs, ADCs, RTC, Battery charger and Fuel gauge

DCDC: TI TPS61259 is 5V step-up DCDC for USB host and HDMI block

Video output: HDMI type-D (Micro-HDMI)

USB: High-speed USB 2.0 host

GPIO connectors: Raspberry Pi-compatible 13x2-pin header on the top side as well as bottom side for 2-way stacking 20+6 pin header for additional GPIO/RTC/USB connection.

GPIOs: A total of 32 GPIOs and 2 ADCs are available.

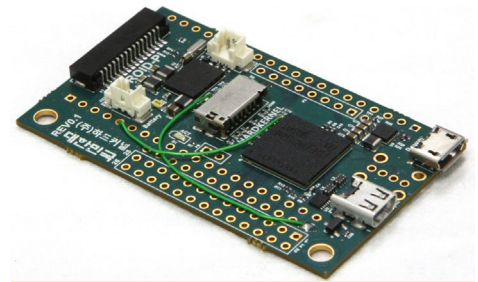
Camera connector: 15pin MIPI-CSI2

Memory card slot: Micro-SD (T-Flash)

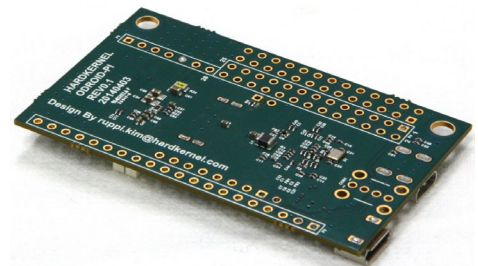
Power: Micro-USB socket for 5V input. Li-Polymer battery connector (Molex 53398-0271)

RTC Power: Backup battery connector (Molex 53398-0271)

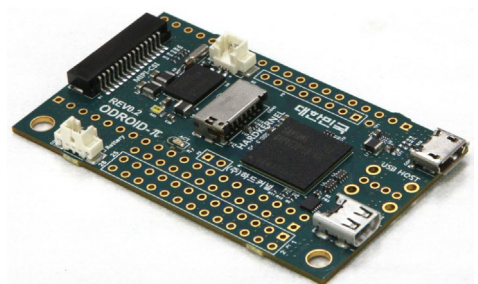
Overall PCB Dimensions: 60 x 36 mm



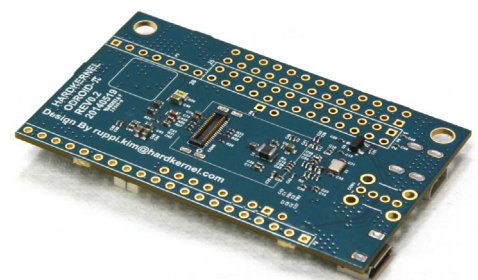
Top view of Rev. 1 ODROID-W board



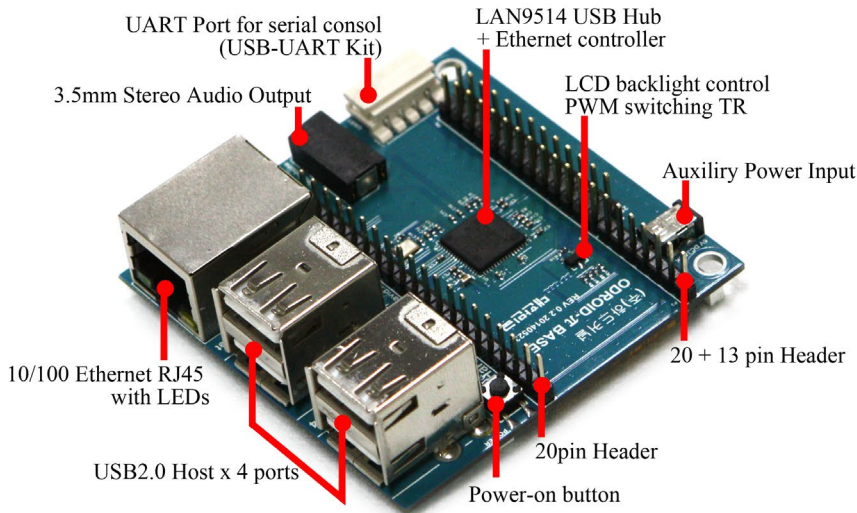
Bottom view of Rev. 1 ODROID-W board



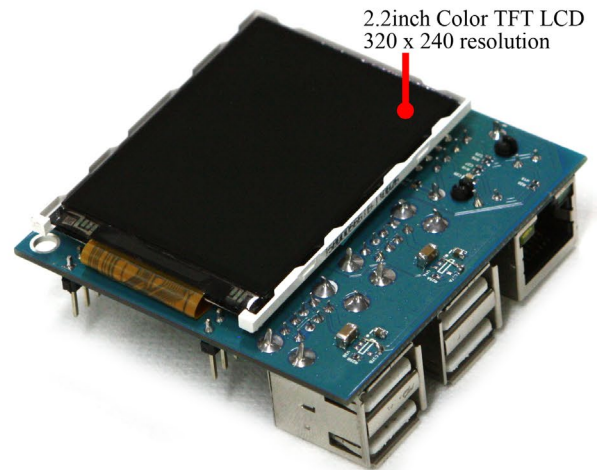
Top view of Rev. 2 ODROID-W board



Bottom view of Rev. 2 ODROID-W board



W Docking Board with LCD
Size : 60 x 60 x 21mm(Approx)

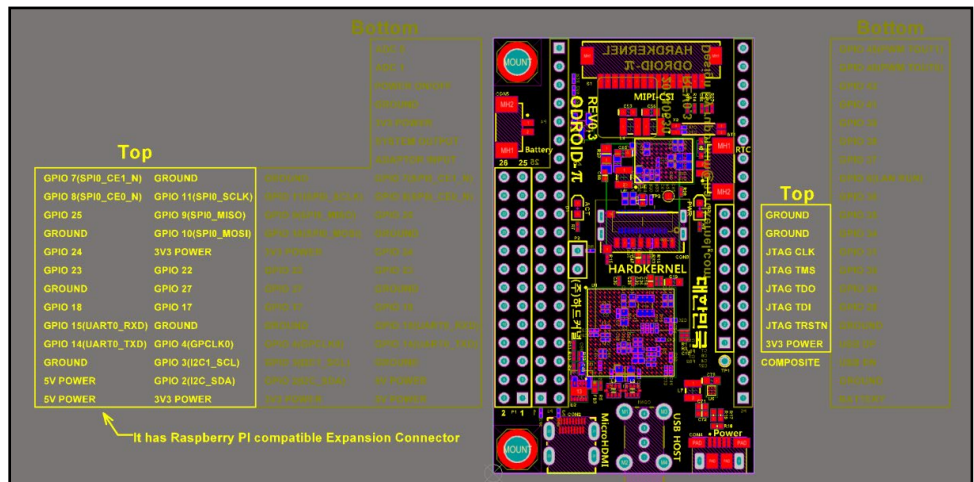


Closeup detail of ODROID-W with Docking Board and LCD

The Docking Board accessory, which attaches to the ODROID-W on either the top or bottom, offers 4 USB host ports and an Ethernet port. You can also choose a 320x240 QVGA TFT LCD-included version.

Please watch our Youtube video of the ODROID-W Docking Board at <http://bit.ly/1rII5Li> for a live demonstration of the board's capabilities. The base price for the Docking Board is \$20, and the TFT LCD-included version is \$30.

The PCB schematic for the ODROID-W is shown to the right for reference.

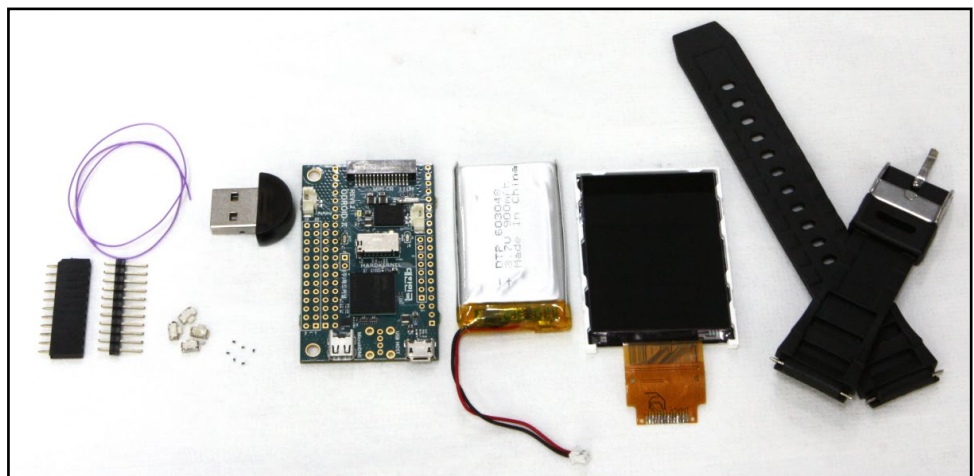


The schematic diagram for the ODROID-W PCB

DIY Smartwatch Example

This DIY smartwatch has some limited but very cool features, and is intended to be tightly coupled with your Android smartphone via Bluetooth.

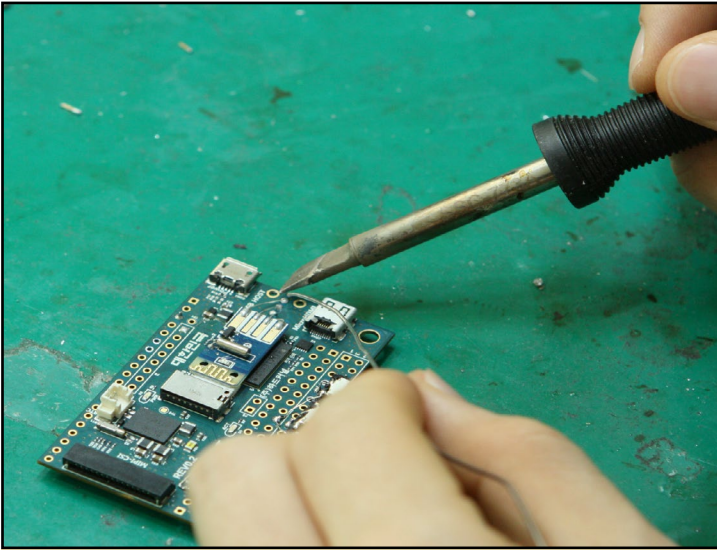
Whenever a message arrives on your phone, a notification is sent to the watch, as seen in our video at <http://bit.ly/1sOkTOC>.



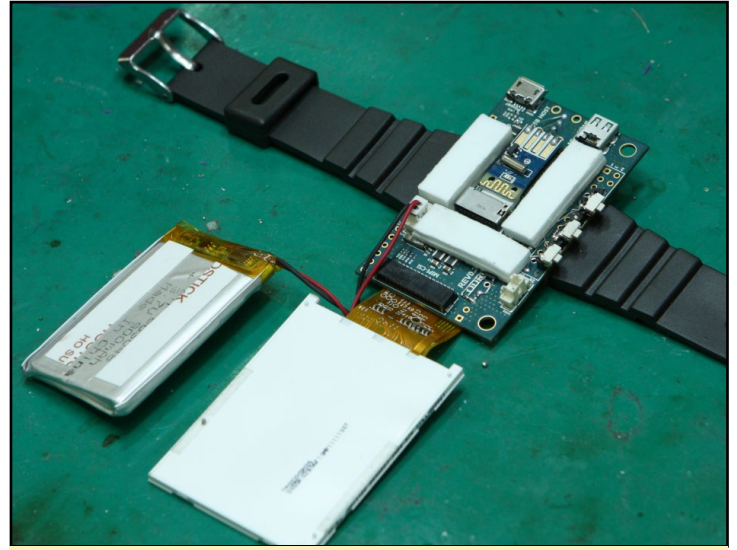
Hardware materials for assembling the Odroid-W smartwatch

The following page illustrates how to assemble your very own fashionable smartwatch from an ODROID-W!
For detailed schematics and software links, please refer to the original post at <http://bit.ly/1rYW8LR>.

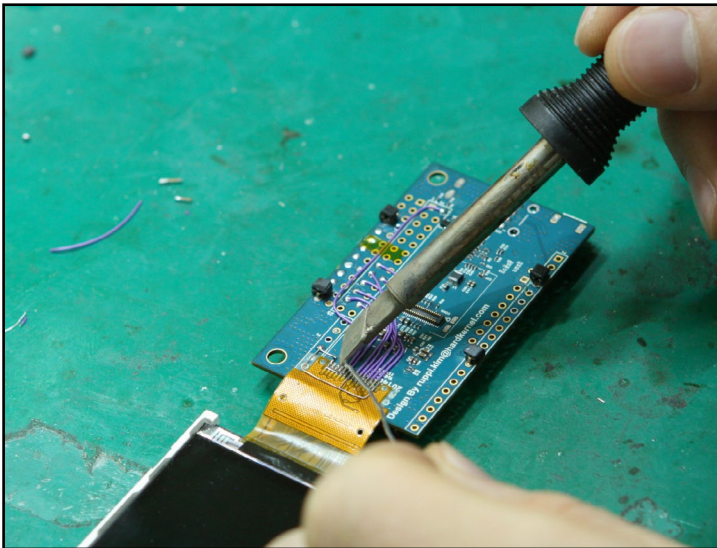
Smartwatch Assembly



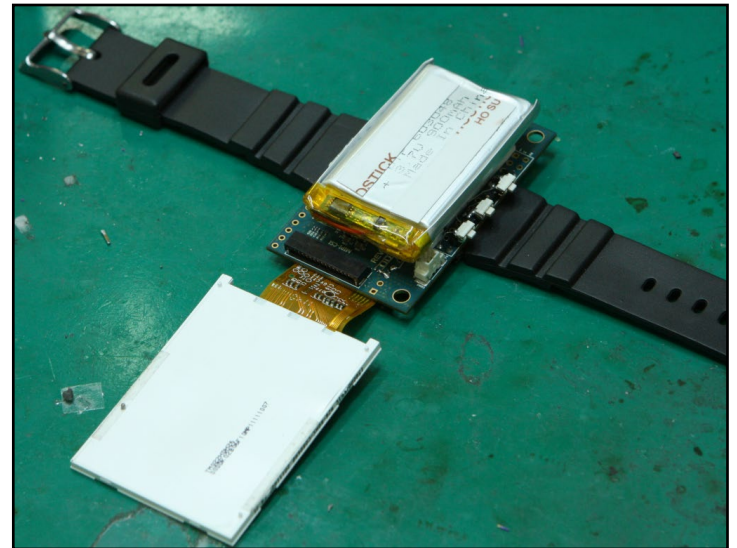
Solder a disassembled USB Bluetooth module and a few buttons.



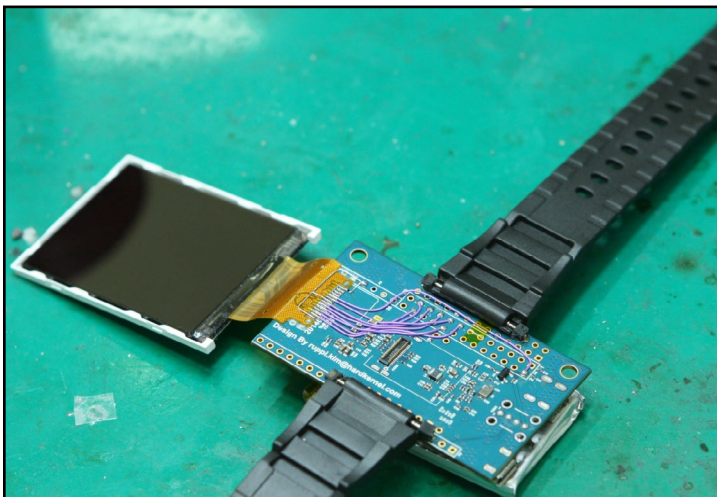
Place double-sided tape on the LCD and on the ODROID-W PCB.



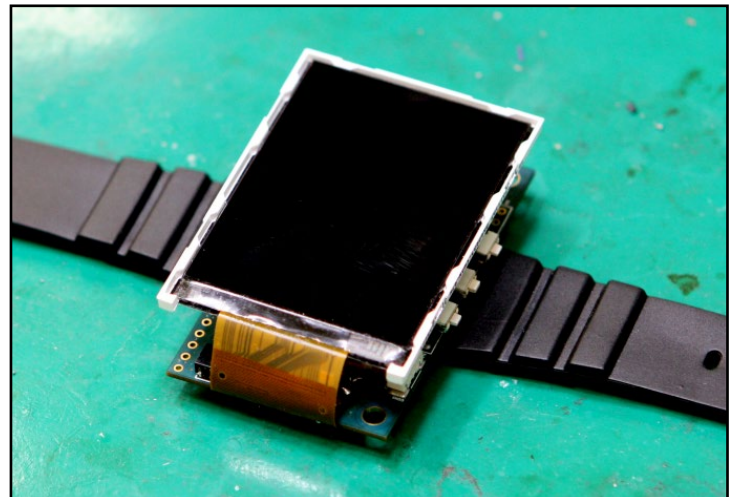
Connect the TFT LCD module with several wires.



Stack the battery and LCD on the ODROID-W PCB with double-side tape as shown above.



Assemble the watch straps. Please note that the hinge holders should be soldered first.



Final assembly of ODROID smartwatch with screen attached.

SEARCH WITH GOOGLE BBS WHAT IF GOOGLE WERE INVENTED IN THE 1980S ?

by Rob Roy

For a retro-style Google experience, visit the Google BBS at <http://bit.ly/1iXk8yF>. Don't forget to bring your moon boots and walkman!



FIXING ANDROID OVERSCAN A SIMPLE APP TO CHANGE THE DESKTOP RESOLUTION

by Rob Roy

Legalabs offers an Android app from the Google Play store that can change the resolution of the desktop in order to fix HDMI monitor overscan issues. For more details, please visit its Google Play Store page at <http://bit.ly/XYfZ51>.

Resolution		
Width		Height
2560		1600
Density		
320		
Overscan		
Top		Right
0		0
Bottom		Left
0		0

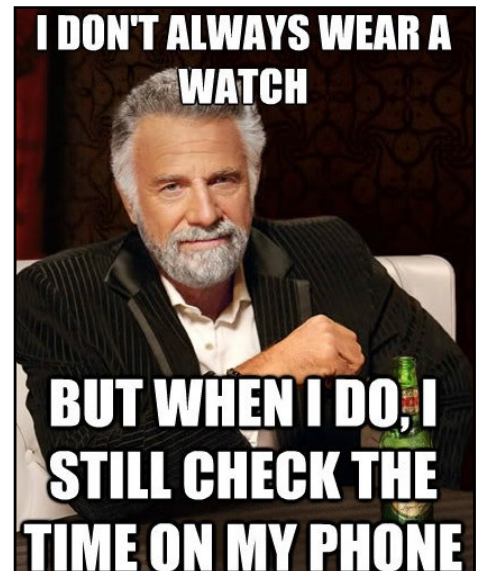
Software

- Debian (Raspbian) OS is running on the Linux Kernel 3.10.
- LXDE X11 window manager (This can be removed if you want)
- Qt 4.8.4 framework library
- Qt application to display the clock and SMS / Mail / Call notification
- Android application software (written in Java) to push the notification to the smartwatch from a smartphone

The full source code will be released via Github at <http://www.github.com/hardkernel>.

Official accessories

- Docking board with TFT LCD
- Docking board without TFT LCD
- Connector package (26-Pin header, 26-Pin header socket, 20-Pin header socket, 7-Pin header socket and USB host right-angle type)
- RTC Backup battery (CR2032 with Molex 51021-0200 wired connector)
- Lithium Polymer battery (3.7V/720mAh with Molex 51021-0200 wired connector)
- Micro-SD 8GB(UHS-1 class) card of Raspbian with RTC/PMIC device driver patch
- RTL8188CUS based USB WiFi module
- 720p USB Webcam



- Micro-HDMI cable
- Micro-USB cable
- 5V/2A Power supply for the Docking board, which is required only when the USB devices need 600mA or higher.
- ODROID-VU (9-inch 1280x800 TFT HDMI monitor with touch screen)

Summary

This tiny version of the Raspberry Pi can run a full Raspbian desktop environment on its own, or be used in conjunction with pre-existing and widely available Pi hardware to create an extremely portable yet powerful computing solution, perfect for wearable applications. Please post any questions that you may have to the ODROID forums at <http://forum.odroid.com>.

I can has smarticles?



To submit your own ODROID-based project for publication, send an email to [odroidmagazine\(at\)gmail.com](mailto:odroidmagazine(at)gmail.com).

ALL ABOUT HARDKERNEL'S EMMC MODULES

THE ODROID ADVANTAGE

by Justin Lee

The Exynos-4412 CPU includes an eMMC 4.41 compatible host controller, with a maximum bus speed limited to 100MB/sec, an actual clock speed of 48Mhz with DDR. The ODROID-X, X2, U2, U3, Q and Q2 are all based on the Exynos-4412 CPU.

The Exynos-5410 CPU, which is used by the the ODROID-XU, XU+E and XU-Lite, includes an eMMC 5.0 compatible host controller. However, the hardware design was unable to support the eMMC 5.0 HS400 mode, because there was no public specification of eMMC 5.0 when we designed the ODROID-XU. As a result, all the XU series have the eMMC 4.5 (HS200) specification which reaches a maximum speed of 160MB/sec since the actual clock speed is 160Mhz with SDR.

The Exynos-5422 CPU also has an eMMC 5.0 compatible host controller. The ODROID-XU3 is based on Exynos-5422 and supports the eMMC HS400 mode. The actual clock is 166Mhz and

330MB/sec of host bus bandwidth by DDR.

We've been selling 2 different eMMC modules: The Green or Blue PCB has the eMMC 4.5 chips.

The Red PCB has the eMMC 5.0 chips. The eMMC 5.0 chip is slightly smaller but it has the same BGA array size, so that we can keep using the same PCB.

Most eMMC manufactures are moving to eMMC 5.0. So it is hard to purchase the eMMC 4.5 chips anymore except for the lower capacity like 4GB and 8GB. Recently, we started using 16GB and 32GB eMMC chips from Sandisk while we keep buying 8GB and 64GB eMMC from Toshiba.

When you install a new OS image onto the eMMC module, you need a USB memory card reader and an eMMC-to-MicroSD converter board (reader board). Please refer to the images below to understand how to use it.

The reader board can't be detected by

a few USB memory card readers. So it is worthwhile to check the compatibility list at <http://bit.ly/1nPBE4i>.

The chart on the following page is the full list of eMMC modules we are selling now. The round label indicates the OS and capacity.

The eMMC 4.5 Blue(or Green) PCBs will be obsolete soon except for 8GB models.

Even you have a different or wrong eMMC, you can use it for your ODROID if you install a proper boot loader and OS image.

Reference

The eMMC board schematics
<http://bit.ly/1p4TX6N>

The reader board schematics
<http://bit.ly/1p9j0Z3>.

The following page is a handy guide to all of Hardkernel eMMC modules!

The eMMC reader attaches to the eMMC module with a small clip



Top view of eMMC reader and eMMC module inserted into SD card reader

































Bottom view of eMMC reader and eMMC module inserted into SD card reader



Hardkernel's removable clip eMMC modules, which are usually soldered into the board, are unique to ODROIDS.

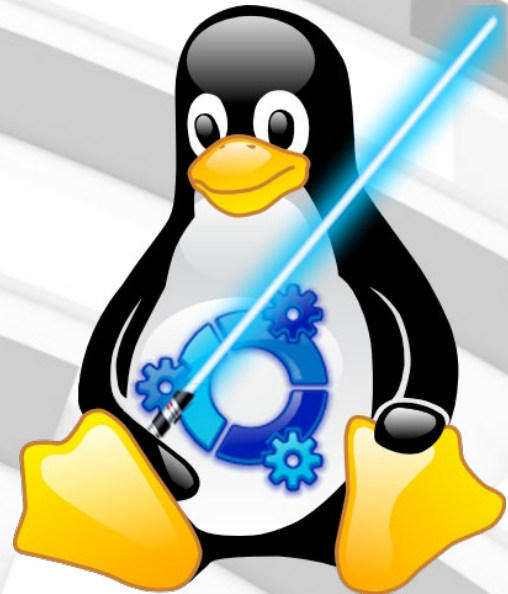
EMMC MODULE REFERENCE CHART

Android for U2/U3 Green						
	8GB	16GB	16GB	32GB	64GB	64GB
Ubuntu For U2/U3 Red						
	8GB	16GB	16GB	32GB	64GB	64GB
Android For X2 Yellow						
	8GB	16GB	16GB	32GB	64GB	64GB
Android For XU Blue						
	8GB	16GB	16GB	32GB	64GB	64GB
Ubuntu for XU3 Light Blue						
			16GB	32GB		64GB
Android For XU3 White						
			16GB	32GB		64GB

LINUX KERNEL COMPILATION

GET FULL CONTROL OF
YOUR OPERATING SYSTEM
LIKE A LINUX JEDI

by Venkat Bommakanti



As you make progress in using Linux, it is inevitable that you would want to tweak your setup in order to accommodate changes in your hardware setup, such as integrating an external device. Such tasks usually require modifications to the Linux kernel or linking to a new driver. This article provides details on starting with natively building a pristine kernel using only the ODROID itself. This article does not address cross-compiling, which relates to building an ODROID kernel using a secondary host machine such as an x86.

Requirements

1. Any ODROID board, with an appropriate power adapter.
2. A bootable 8+ GB MicroSD card or eMMC module containing the latest Ubuntu image available from the Hardkernel website.
3. A network where the device has access to the Internet and the ODROID forums.
4. Optional SSH access to the U3 via utilities like PuTTY (MS Windows 7+) or Terminal (Mac, linux) to perform the steps from a remote host computer.

System basics

Initially, for reference, it is a good idea to note the version of the initial kernel version included with the installed image with the following command:

```
$ uname -a
```

```
Linux u3-2 3.8.13.26 #1 SMP PRE-EMPT Wed Jul 9 22:14:37 UTC 2014  
armv7l armv7l armv7l GNU/Linux
```

This output indicates the image used is based on the 3.8.y.z kernel, built on Wed Jul 9 22:14:37 UTC 2014.

You can get additional image information using the following commands and observing the output:

```
$ cat /proc/version
```

```
Linux version 3.8.13.26 (root@THEserver) (gcc version 4.7.2  
(crosstool-NG 1.17.0) ) #1 SMP PREEMPT Wed Jul 9 22:14:37 UTC  
2014
```

```
$ cat /etc/lsb-release
```

```
DISTRIB_ID=Ubuntu  
DISTRIB_RELEASE=14.04  
DISTRIB_CODENAME=trusty  
DISTRIB_DESCRIPTION="Ubuntu 14.04  
LTS"
```

```
$ lsb_release -a
```

```
No LSB modules are available.  
Distributor ID: Ubuntu  
Description: Ubuntu 14.04  
LTS  
Release: 14.04  
Codename: trusty
```

```
$ cat /etc/os-release
```

```
NAME="Ubuntu"  
VERSION="14.04, Trusty Tahr"  
ID=ubuntu  
ID_LIKE=debian  
PRETTY_NAME="Ubuntu 14.04 LTS"  
VERSION_ID="14.04"  
HOME_URL="http://www.ubuntu.com/"  
SUPPORT_URL="http://help.ubuntu.com/"  
BUG_REPORT_URL="http://bugs.launchpad.net/ubuntu/"
```

After installing a newly built kernel, the above information is useful for comparison when verifying that the kernel has actually been updated.

Preparing the system

Prior to building a kernel, you would need to setup the appropriate software environment. First, launch a terminal session and switch to the root user, using the command:

```
$ sudo -s
```

All build related activities need to be performed by the root user, which can be enabled with the `su` command after typing the root password.

Install the initial set of needed components, using the following installation command:

```
# apt-get install build-essential
libqt4-dev perl python git
```

Enter Y to accept proposed option, and allow for the command to complete.

This is a comprehensive version of the command, highlighting all components required for building a kernel. Some of the required components may already be installed in your present image, which will not be affected by the above command since only missing components will be installed.

The installation output may suggest additional optional components. Although not mandatory, you may install those using the component list, like this:

```
# apt-get install nas libqt4-
declarative-folderlistmodel
libqt4-declarative-gestures
libqt4-declarative-particles
libqt4-declarative-shaders
```

For desktop images, presuming the default window-system backend is based on X11/Xorg, you should install libxcb and its accompanying packages. This is also used for building QT4. It is installed using the command:

```
# sudo apt-get install "^libx-
cb.*" libx11-xcb-dev libglul-me-
sa-dev libxrender-dev
```

Enter Y to accept proposed option. QT4 support may not exist on Hardkernel images by default.

Check the system

It is essential to make sure the system is ready for compilation. In this example case, the following commands/outputs are useful to verify that all of the tools are available and up to date:

```
# gcc --version
gcc-4.8.real (Ubuntu/Linaro
4.8.2-19ubuntu1) 4.8.2
Copyright (C) 2013 Free Software
Foundation, Inc.
This is free software; see the
source for copying conditions.
There is NO
warranty; not even for MERCHANT-
ABILITY or FITNESS FOR A PARTICU-
LAR PURPOSE.

# pkg-config --modversion QtCore
4.8.6

# perl --version

This is perl 5, version 18, sub-
version 2 (v5.18.2) built for
arm-linux-gnueabi-hf-thread-multi-
64int
(with 41 registered patches, see
perl -V for more detail)
Copyright 1987-2013, Larry Wall
...

# python --version
Python 2.7.6
```

```
# git --version
git version 1.9.1
```

Fetch the source code

Create a temporary directory in your home directory and navigate to it using the commands:

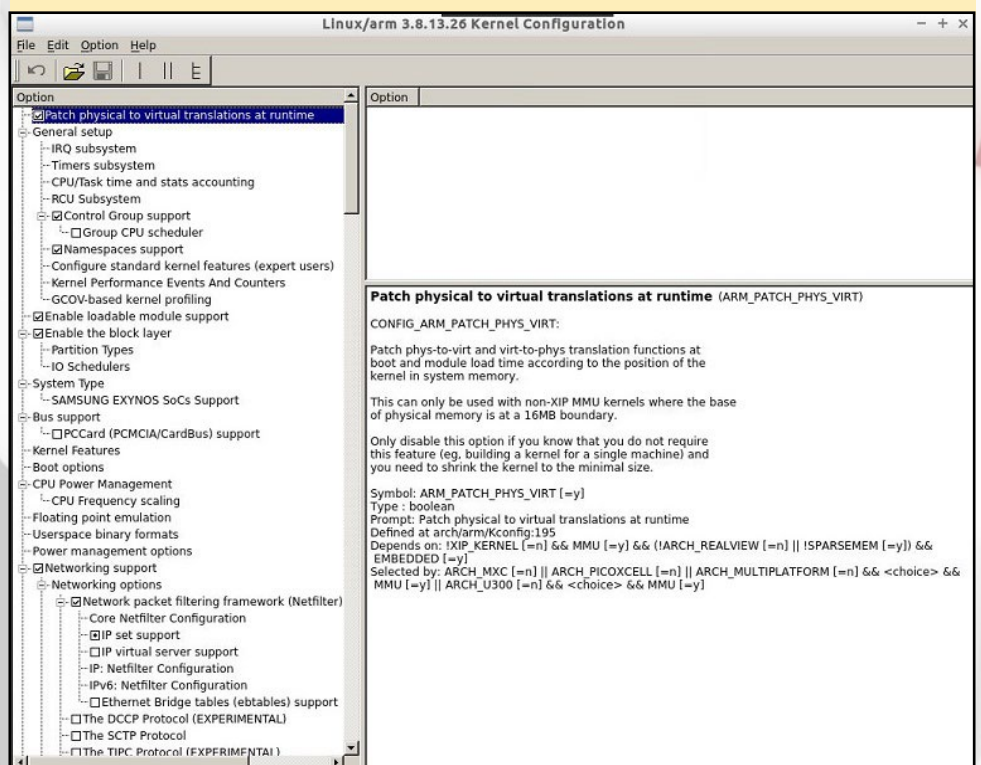
```
# mkdir hk-src
# cd hk-src
```

Your copy of the downloaded source code will reside here. Hardkernel uses GitHub for the source-control of its open-source software for U3.

Use the following command to check out the 3.8 repository, which is recommended for the X2 and U3:

```
# git clone --depth 1 https://
github.com/hardkernel/linux.git
-b odroid-3.8.y odroid-3.8.y
Cloning into 'odroid-3.8.y'...
remote: Counting objects: 44530,
done.
```

Kernel configuration options are available for experienced users to add and change drivers



```
remote: Compressing objects: 100%
(42780/42780), done.
remote: Total 44530 (delta 3428),
reused 16016 (delta 1191)
Receiving objects: 100%
(44530/44530), 122.03 MiB |
161.00 KiB/s, done.
Resolving deltas: 100%
(3428/3428), done.
Checking connectivity... done.
Checking out files: 100%
(42143/42143), done.
```

In this case, we will fetch a cloned copy of only the latest commit to the 3.8.y kernel, instead the entire repository. Please note that if you are using an ODROID-XU, clone the 3.4.y branch instead:

```
# git clone --depth 1 https://
github.com/hardkernel/linux.git
-b odroid-3.4.y odroid-3.4.y
```

Select a defconfig

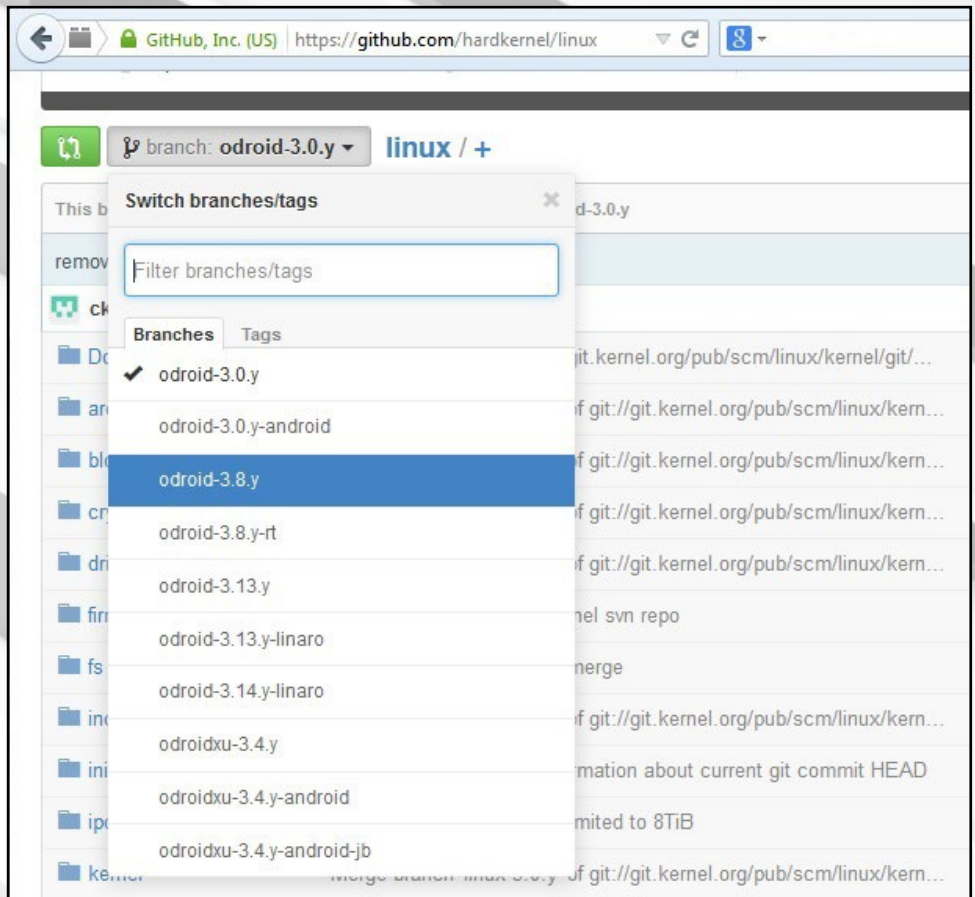
Change to the source root directory using the command:

```
# cd odroid-3.8.y/
```

This is where all build related activities are launched from. Check the list of supported defconfig files using the command:

```
# ls -lsa arch/arm/configs/
odroid*defconfig
100 -rw-r--r-- 1 root root 101207
Jul 14 13:11 odroidu_defconfig
96 -rw-r--r-- 1 root root 96792
Jul 14 13:11 odroidx2_defconfig
96 -rw-r--r-- 1 root root 96797
Jul 14 13:11 odroidx_defconfig
```

In our case, we will select the odroidu_defconfig version, which contains hardware configuration options specific to the U3. Now, perform the first build step by executing the follow-



Hardkernel maintains open-source kernel repositories that are downloadable for free.

ing command, ignoring any warnings:

```
root@u3-2:~/odroid-3.8.y# make
odroidu_defconfig
HOSTCC scripts/basic/fixdep
...
#
# configuration written to .config
#
```

Configure additional modules

In this example case, since QT4 support was added (xconfig is a QT4 UI), the xconfig module needs to be built using the following command to launch an editor for the selection of options:

```
# make xconfig
CHECK qt
MOC scripts/kconfig/qconf.moc
HOSTCXX scripts/kconfig/qconf.o
HOSTLD scripts/kconfig/qconf
```

Novice users should review and go through the options list but not alter any selections. Experienced users may want to adjust the selections in the kernel configuration menu based on their in-depth knowledge of the kernel.

Baseline resource usage

To understand the implications of the build process on resource usage, it is useful to get a picture of the resource usage, before and during, the build process, as shown in the screenshot of the command `top`.

Build the kernel and selected modules

To compile the kernel as fast as possible, using all 4 processors available in the U3, type the following command in the Terminal window:

```
# top
top - 13:35:15 up 2:33, 4 users, load average: 0.09, 0.13, 0.14
Tasks: 142 total, 1 running, 141 sleeping, 0 stopped, 0 zombie
%Cpu(s): 2.9 us, 0.7 sy, 0.0 ni, 96.3 id, 0.0 wa, 0.0 hi, 0.1 si, 0.0 st
KiB Mem: 2071392 total, 1519108 used, 552284 free, 78556 buffers
KiB Swap: 0 total, 0 used, 0 free. 1206844 cached Mem

  PID USER      PR  NI  VIRT  RES  SHR  S  %CPU  %MEM  TIME+  COMMAND
  929 root        20   0  88388 49212 34172 S  10.9  2.4   3:45.61 Xorg
 1066 root        20   0  38852 26304 14756 S   3.0  1.3   1:34.00 xllvnc
 4801 odroid     20   0   2572  1184   764 R   0.7  0.1   0:00.07 top
 1276 root        20   0     0     0     0 S   0.3  0.0   0:00.23 kworker/0:2
 1614 odroid     20   0 202540 15084 9492 S   0.3  0.7   0:17.28 lxterminal
 4437 root        20   0     0     0     0 S   0.3  0.0   0:01.77 kworker/u:1
     1 root        20   0   3532  2160  1100 S   0.0  0.1   0:04.60 init
...
```

Resource usage before build starts

```
# make -j5 zImage modules
```

It is recommended to perform the build process on an ODROID that uses active fan cooling. You can experiment with higher -j values only if you are able to keep the temperature of the ODROID below the thermal limits. For the above command, the build process may take 15 ~ 30 mins, depending on pre-existing system activity. Pay attention to build-time warnings, especially related to your changes, if any.

It is interesting the note the very high cpu-core usage (~90%) during the build process, as seen in the above figure. As expected, the fan was spinning non-stop!

Install kernel and modules

The freshly built kernel and modules can be installed using the following commands:

```
# cp arch/arm/boot/zImage /media/
boot/zImage
```

```
# top
top - 13:38:36 up 2:37, 4 users, load average: 5.59, 2.45, 1.00
Tasks: 164 total, 6 running, 158 sleeping, 0 stopped, 0 zombie
%Cpu0 : 89.1 us, 8.9 sy, 0.0 ni, 0.3 id, 0.0 wa, 0.0 hi, 1.7 si, 0.0 st
%Cpu1 : 94.7 us, 5.3 sy, 0.0 ni, 0.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
%Cpu2 : 93.4 us, 6.6 sy, 0.0 ni, 0.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
%Cpu3 : 91.4 us, 8.6 sy, 0.0 ni, 0.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
KiB Mem: 2071392 total, 1612744 used, 458648 free, 83180 buffers
KiB Swap: 0 total, 0 used, 0 free. 1227700 cached Mem
```

Resource usage (top with P option) during build process

```
# make modules_install
```

Update initramfs

Prior to boot-up using the new kernel, the following steps need to be performed to create the initial RAM filesystem (initramfs). First, copy the current config to /boot so update-initramfs can run properly, using the command:

```
# cp .config /boot/config-`cat include/config/kernel.release`
# ls -lsa /boot/conf*
100 -rwxrwxrwx 1 root root
101420 Jun 13 01:02 /boot/config-3.8.13.23
100 -rwxrwxrwx 1 root root
101207 Jul 14 14:15 /boot/config-3.8.13.26
```

Recreate the initramfs using the command:

```
# update-initramfs -c -k `cat \
```

```
include/config/kernel.release`
update-initramfs: Generating /
boot/initrd.img-3.8.13.26
```

Convert the current initrd into a u-boot compatible version, which adds a 64byte u-boot header:

```
# mkimage -A arm -O linux -T ram-
disk -C none -a 0 -e 0 -n uInitrd
-d /boot/initrd.img-`cat include/
config/kernel.release` /boot/uIni-
trd-`cat include/config/kernel.
release`
Image Name: uInitrd
Created: Mon Jul 14 14:17:02
2014
Image Type: ARM Linux RAMDisk
Image (uncompressed)
Data Size: 2196336 Bytes =
2144.86 kB = 2.09 MB
Load Address: 00000000
Entry Point: 00000000
```

Save the current uInitrd using the command:

```
# cp /media/boot/uInitrd /boot/
uInitrd-`uname -r`
# ls -lsa /boot/u*
2144 -rwxrwxrwx 1 root root
2195134 Jul 13 22:43 /boot/uIni-
trd
2144 -rwxrwxrwx 1 root root
2195134 Jul 14 14:17 /boot/uIni-
trd-3.8.13.26
```

Enable the new uInitrd using the command:

YOUTUBE PLAYER ALTERNATIVE

USE TAMPERMONKEY TO WATCH VIDEOS

by Jeremy "Cartridge" Kenney



```
# cp /boot/uInitrd-`cat include/
config/kernel.release` /media/
boot/uInitrd
# ls -lsa /boot/u*
2144 -rwxrwxrwx 1 root root
2195134 Jul 14 14:18 /boot/uIni-
trd
2144 -rwxrwxrwx 1 root root
2195134 Jul 14 14:17 /boot/uIni-
trd-3.8.13.26
```

Reboot

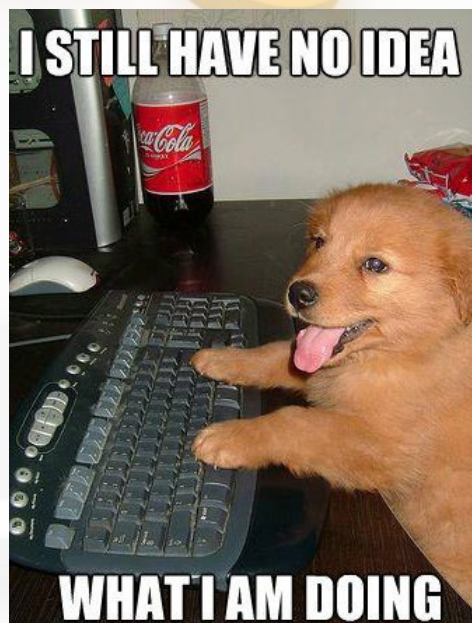
If no errors have been encountered, reboot the system using the following commands so that the new kernel may take effect:

```
# sync && reboot
```

After the reboot has completed, you can verify that the new kernel is installed using the `uname` command to compare the timestamp and kernel version from before and after the build:

```
# uname -a
Linux u3-2 3.8.13.26 #1 SMP PRE-
EMPT Mon Jul 14 14:02:33 PDT 2014
armv7l armv7l armv7l GNU/Linux
```

For questions, please visit the original forum post at <http://bit.ly/1ucMAIN>.



Is your Youtube defaulting to the HTML5 player, and any Chrome Web-Store addon that you've tried does not work for forcing Flash Player instead? Are you tired of not using the resolution you want on Youtube or the bland white look of Youtube's Skin? Knowing a trick or two about Youtube can help you address these issues, and even give you the option for a full screen or windowed version of your video clip for faster multi-tasking.

Here's a neat simple trick using Tampermonkey and a short script!

Tampermonkey is like Greasemonkey (Firefox) but for Chrome. And for those who don't know what

GreaseMonkey is, it's an extension that injects scripts into the website you are using and other neat features using scripts.

You can go to Chrome's Webstore and grab the Extension from <http://bit.ly/1iL5YRd>. You can use scripts on any website that does not have a security against scripts.

There's endless possibilities for developers and an easy interface for beginners that allows for quick and easy management of your scripts.

If you are using an earlier build of Chromium for use on early Debian Builds, install the Legacy Tampermonkey instead. This requires you to Download the CRX from this website

Tampermonkey is available from the Chrome Web Store


Download
Features

Tampermonkey is a free browser extension and the most popular userscript manager for Blink-based Browsers like Chrome and Opera Next. Even though Google Chrome does have native support for userscripts, Tampermonkey can give you much more convenience in managing your userscripts. It provides features like easy script installation, automatic update checks, an overview what scripts are running at a tab and there is a good chance that scripts that are incompatible to Google Chrome run with Tampermonkey.

Tampermonkey is available for Google Chrome, Opera, Chromium and a lot of their derivatives like CoolNovo and Rockmelt and it is installed in just a minute, so give it a try!


+1 | 163
f Like
Share | 4k

Download/Installation




Tampermonkey Stable

Chrome >= 22




Tampermonkey Beta

Chrome >= 22



Tampermonkey Legacy

Chrome >= 17
Chrome < 27



Tampermonkey Android

Android >= 2.2

Download Mozilla Firefox

mozilla-firefox.onfreedownload.com
 Official 2014 Version for MacOS.
 Hurry Up Right Now: Free Download!

>

INTERESTING LINUX COMMANDS CUTE PROGRAMS FOR YOUR NEXT COFFEE BREAK

by Rob Roy

Want to see some of the fun side of Linux? Try these commands:

All aboard!

```
$ sudo apt-get install sl
$ sl
```

Predict the future

```
$ sudo apt-get install \
fortune-mod
$ fortune
```

Moo

```
$ sudo apt-get install cowsay
$ cowsay 'ODROIDS are cool!'
```

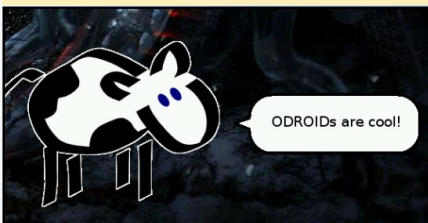
Cow fortune teller

```
$ sudo apt-get install \
cowsay \
fortune-mod
$ fortune | cowsay
```

XII cow divination

```
$ sudo apt-get install \
xcowsay \
fortune-mod
$ fortune | xcowsay
```

The XII cow is always right!



at <http://tampermonkey.net>. To install, Right-click and Save As to the folder of your choice, then open up Chromium settings and click on the Extension tab. Drag 'n' Drop the CRX file you have downloaded from the link above onto the Extension Tab Page. Finally, test out the new options by going to <http://www.youtube.com> and watching a video.

Youtube Center

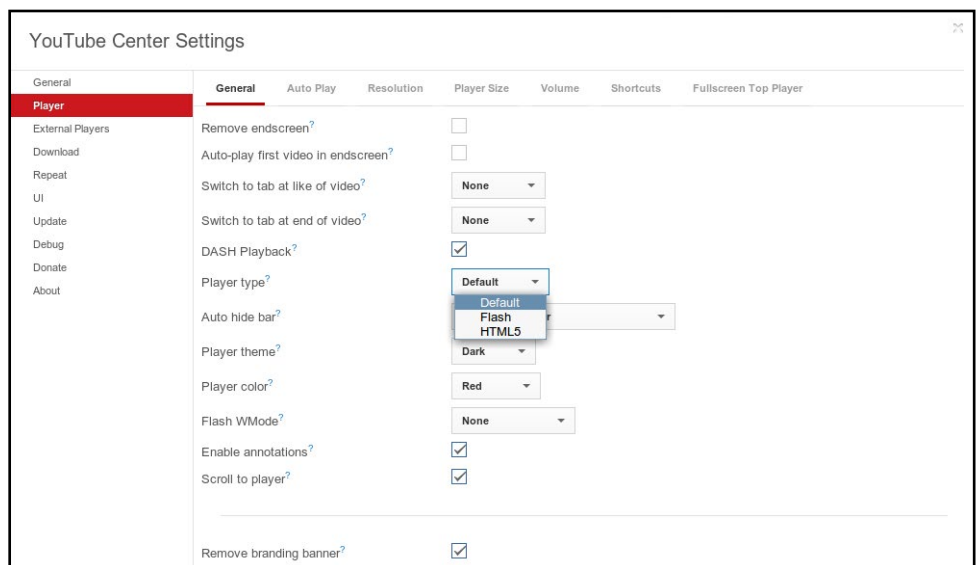
This script gives you access to options you may have never seen before on Youtube. You get to make use of Auto

Click install, then open up your Youtube and a window will prompt you to show you where the new settings are.

Click on the Player tab and enable Flash Playback in Player Type. If you have a fast connection, disable Dash Playback for faster loading and fix various hiccups made by a weak wifi signal.

You now have access to more options you can think of, and can customize Youtube in any way you like, even changing the player theme and color.

The script can be disabled by clicking on the Tampermonkey button in the extension bar beside the Address bar, then



Changing the player settings in the Youtube Center

Resolution to adjust the video to the resolution of your choice, set the width and height of the windowed player, enable or disable use of Dash Playback (which is block buffering instead of loading the full clip for safe use on 3G).

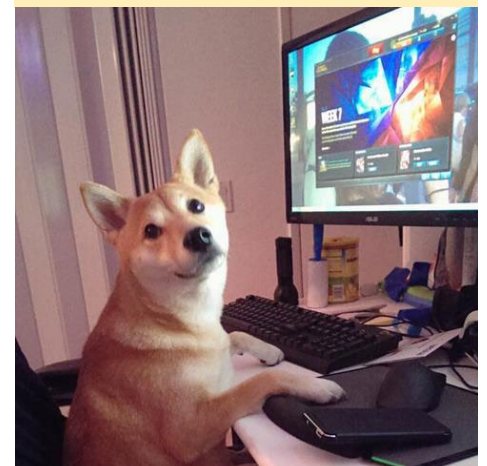
You can also add a download button on each video you watch to select the format of your choice, add a repeat button if you like that song you're playing, and many more features.

One of the more useful options is the HTML5 Alternative, Flash Playback. Go to Youtube Center's Github at <http://bit.ly/MANHYG>, scroll down to Download and click Dropbox. Your Tampermonkey should open up right away and prompt you if you want to install the script you have chosen.

clicking Dashboard and unchecking the extensions of your choice.

Have fun enjoying YouTube!

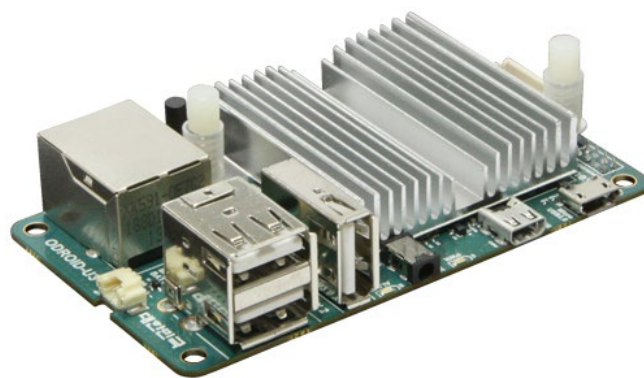
We got YouTube working, and now all he does is upload videos of his expert gaming skills.



ODROID-U3 VS ODROID-U3+

THE NEXT GENERATION IN THE ODROID-U SERIES

by Justin Lee



The U3+ improves upon the popular U3 by adding another USB port via host mode

We've been shipping the ODROID-U3+ (Rev 0.5) since the end of June 2014, with 4 main improvements over the original U3:

- Micro-USB port is able to work in USB host mode.
- The USB host power protector IC was changed from AP2411 to NCP380.
- HDMI reverse-current blocking IC AP2331 was placed on the correct place.
- HW SPI port was added.

USB host mode with Micro-USB port

By adding a power control IC RT9715, the Micro-USB port can work in the USB host mode as well as in USB device mode, which we call Dual-Role-Device (DRD). Just plug in an OTG-to-Host cable and you will have another USB host available for peripherals, so that you can enjoy 4 USB ports without



OTG-to-Host cable to allow use of the Micro-USB port as a 4th USB host port

using an external bulky USB hub. It's a dual-use port that works both ways!

New USB host power protector IC NCP380

AP2411 was used in Rev 0.2 PCB as a USB bus power protector. However, some users reported that it could be damaged by an electrical shock on the USB port.

To solve this issue, we removed the chip and added a wire, which changed it to more durable NCP380 that includes many reliable features like Under voltage lock-out, Built-in Soft start, Thermal Protection, Soft turn-off, Reverse voltage Protection, ESD Compliance to IEC61000-4-2 (Level 4), 8.0 kV (Contact), and 15 kV (Air).

HDMI reverse-current blocking IC

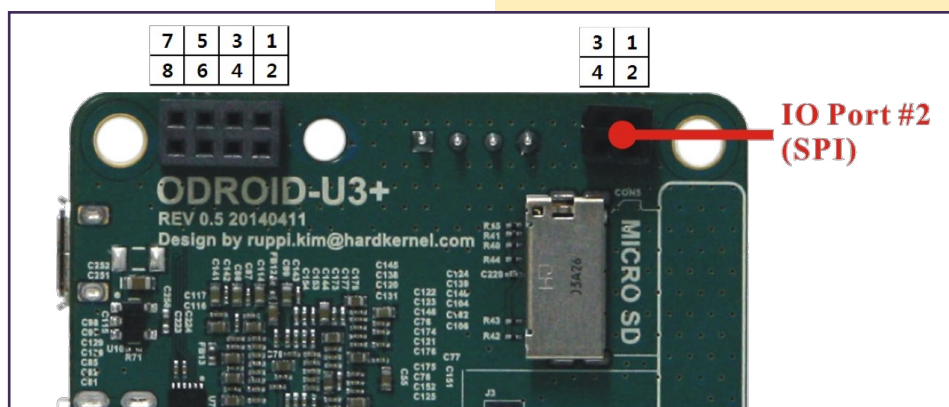
Some monitors or TVs have leak-

age current on the HDMI port. It puts the PMIC into an undefined state and it prevents the auto-power-on function. We moved the reverse current protection IC from HDMI connector side to CPU/PMIC side, which solved the problematic auto-power-on issue. Now we can turn on the board automatically by simply inserting the DC power jack even when the HDMI is connected.

HW SPI port

The Serial Peripheral Interface (SPI) bus is a synchronous serial data link that operates in full duplex mode. The SPI bus is added on the new J5 (IO Port #2) 4-pin connector on the PCB. Note that SPI ports support only master mode and a 1.8V interface like the other IO ports on Exynos processor. You can use 4 pins as GPIO mode as well as SPI mode. After booting, the pins are in GPIO mode by default.

Pin map bottom side view



Pin Number	SPI Function	CPU GPIO pin	Node Number
1	SCLK	GPB[4]	20
2	nSS	GPB[5]	21
3	MOSI	GPB[7]	23
4	MISO	GPB[6]	22

Pin Descriptions

This example script sets all pins to output and toggle 5 times at 1Hz frequency. At the beginning of the application launch, it unloads the SPI driver modules to make sure the GPIO mode, and releases the GPIOs for the next usage at the end.

```
modprobe spi-s3c64xx
modprobe odroid-ioboard
dmesg | grep ioboard
modprobe -r odroid-ioboard
modprobe -r spi-s3c64xx

count=0
stop=5

echo 20 > /sys/class/gpio/export
echo 21 > /sys/class/gpio/export
echo 22 > /sys/class/gpio/export
echo 23 > /sys/class/gpio/export

echo out > /sys/class/gpio/
gpio20/direction
echo out > /sys/class/gpio/
gpio21/direction
echo out > /sys/class/gpio/
gpio22/direction
echo out > /sys/class/gpio/
gpio23/direction

while :
do
count=$((count+1))

echo 1 > /sys/class/gpio/gpio20/
value;
echo 1 > /sys/class/gpio/gpio21/
value;
echo 1 > /sys/class/gpio/gpio22/
value;
```

Chart of U3+ GPIO pin arrangements

```
echo 1 > /sys/class/gpio/gpio23/
value;
echo "GPB[4],GPB[5],GPB[6],GPB[7]
set output HIGH"
sleep 1;

echo 0 > /sys/class/gpio/gpio20/
value;
echo 0 > /sys/class/gpio/gpio21/
value;
echo 0 > /sys/class/gpio/gpio22/
value;
echo 0 > /sys/class/gpio/gpio23/
value;
echo "GPB[4],GPB[5],GPB[6],GPB[7]
set output LOW"
sleep 1;

if [ $count -eq $stop ]; then
echo "GPIO TEST STOP"
echo 20 > /sys/class/gpio/unex-
port
echo 21 > /sys/class/gpio/unex-
port
echo 22 > /sys/class/gpio/unex-
port
echo 23 > /sys/class/gpio/unex-
port
exit 0
fi
done
```

Let's access the SPI mode. If the version of your kernel was downloaded before July 17, 2014, you must first update the kernel first.

SPI driver module

For generic SPI usage, first load the module to activate the SPI host.

```
$ sudo modprobe spi-s3c64xx
```

Then, load the module to activate the generic SPI, and you will have a standard SPI node.

```
$ sudo modprobe spidev
$ /dev/spidev1.0
```

The procedure for Serial Flash also begins by loading the module to activate the SPI host.

```
$ sudo modprobe spi-s3c64xx
```

Then, load the serial flash (misc) driver, and you will have a Serial Flash node.

```
$ sudo modprobe odroid-ioboard
$ /dev/ioboard-spi-misc
```

Please note that spidev.ko and odroid-ioboard.ko can't be loaded at the same time.

SPI host interface driver:
<http://bit.ly/1qz8gHs>

SPI generic driver
<http://bit.ly/1rILc5X>

Serial SPI Flash add-on driver
example <http://bit.ly/1sOpccA>

We tested a Serial SPI Flash memory SST25WF020A on the new IO Shield, and the maximum speed was 40Mhz of SPI clock. To learn more about the SPI-DEV driver in detail, please visit <http://bit.ly/WAgUYC>.

The ODROID-U3 Rev 0.5 has some great hardware improvements, so we called it the ODROID-U3+. You can find the latest U3 Rev 0.5 schematics here at <http://bit.ly/1oUSHMn>. In addition, the new U3 IO Shield Rev 0.3 has been improved to include a new SPI Flash memory which can be connected to an SPI bus (<http://bit.ly/1nSPTFj>).

INSTALL A HOME WEB SERVER

USING LIGHTTPD AND NGINX TO PUBLISH YOUR WEBSITES

by @hamiko and Venkat Bommakanti

One of the most common uses for a Linux machine is to create a web server for the purposes of hosting a website, which is an essential component to any business strategy or personal project. Whether you want to create a live production server or a personal one for your home intranet, this article describes the steps involved in enabling two of the popular light-weight web servers: nginx & lighttpd (aka lighty).

Although Apache server is also a popular choice for Linux web hosting, it is covered extensively on the PHP website at <http://bit.ly/1wXDju1> with specific instructions for the ODROID located at <http://bit.ly/1rIb9Te>.

The instructions also include adding support for a secure web server (openssl-based HTTPS) and scriptable web applications (PHP5-FPM). FPM, which stands for FastCGI Process Manager, is an efficient alternative to the classic FastCGI implementation. If you are concerned about security related issues and configuration when hosting a website, it is recommended to research it well and implement relevant procedures before enabling access to the Internet. Periodic OS updates are also recommended.

Requirements

1. Any ODROID board, with an appropriate power adapter.
2. A bootable 8+ GB MicroSD card

or eMMC module containing the latest U3 Lubuntu image available from the Hardkernel website.

3. A network where the device has access to the Internet and the ODROID forums.

4. Optional SSH access to the U3 via utilities like PuTTY (MS Windows 7+) or Terminal (Mac, linux to perform the steps from a remote host computer.

Server Installation

Although it is possible to install both web servers on the U3 at the same time, the steps below assume that only one web server is being installed. For simultaneous use, any configurations conflicts, such as the use of the default web port 80 by both servers, should be resolved first.



Web designers

INTERESTING LINUX COMMANDS CUTE PROGRAMS FOR YOUR NEXT COFFEE BREAK

by Rob Roy

Want to see some more of the fun side of Linux? Try these commands:

Odd name

```
$ sudo apt-get install toilet
$ toilet -f mono12 -F metal \
  ODROID Magazine
```



The toilet command thankfully does not do what you think it does!

Enter the matrix

```
$ sudo apt-get install cmatrix
$ cmatrix
```

Chase the mouse

```
$ sudo apt-get install oneko
$ oneko
```

Do you want to play a game?

```
$ sudo apt-get install espeak
$ espeak \
  'Do you want to play a game?'
```

Cozy warm fire

```
$ sudo apt-get install libaa-bin
$ aafire
```

1 - Lighttpd and Nginx

Setup root rights, by running the command `sudo -s`

2 - Nginx

Install nginx using the command:

```
# sudo apt-get install nginx
```

Launch nginx using the commands:

```
# sudo service nginx stop
# sudo service nginx start
```

2 - Lighttpd

Install lighttpd using the command:

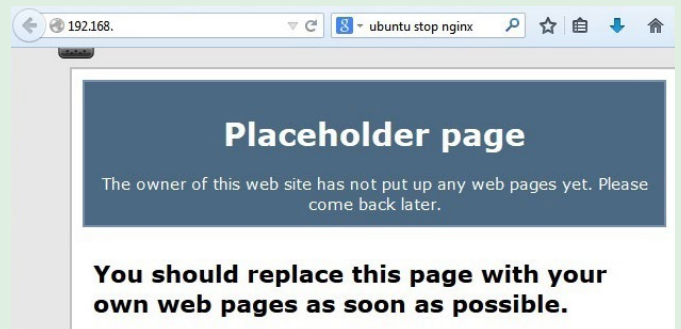
```
# sudo apt-get install lighttpd
```

Launch lighttpd using the commands:

```
# sudo service lighttpd stop
# sudo service lighttpd start
```

3 - Nginx and Lighttpd

Launch a web-browser such as firefox and point to the ip-address of the device, using the URL `http://<u3-ip-address>`



4 - Nginx

Install self-generated ssl keys/certificates, using the following (single-line) commands (use command help to get details on specified options):

```
# mkdir /etc/nginx/ssl
# sudo openssl req -x509 -nodes -days 365 \
-newkey rsa:2048 -keyout \
/etc/nginx/ssl/nginx.key \
-out /etc/nginx/ssl/nginx.crt
```

Appropriate destination directories first need to be created, if not already present.

When prompted for certificate information, you can enter information such as:

```
Country Name (2 letter code) [AU]:US
State or Province Name [Some-State]:CA
Locality Name (eg, city):San Jose
```

4 - Lighttpd

Check if SSL has been enabled in the lighttpd application, using the command:

```
# lighttpd -v
lighttpd/1.4.33 (ssl) - a light and fast webserver
Build-Date: Jan 28 2014 17:19:37
```

Note that the output reflects that SSL is enabled. Now, install self-generated ssl keys/certificates, using the following (single-line) commands:

```
# mkdir /etc/lighttpd/ssl
# sudo openssl req -x509 -nodes -days 365 \
-newkey rsa:2048 -keyout \
/etc/lighttpd/ssl/lighttpd.key \
-out /etc/lighttpd/ssl/lighttpd.crt
```

Appropriate destination directories need to be created first, if not already present.

4 - Nginx

```
Organization Name:YourCompany
Organizational Unit Name:Engineering
Common Name:YourName
Email Address:you@yourcompany.com
```

Check generated certificate, using the command:

```
# openssl s_client -connect localhost:443 \
-reconnect
```

If the certificate information entered earlier is now emitted, then it can be assumed the certificate was generated properly.

Add SSL configuration to the nginx configuration, using the following steps:

```
# cd /etc/nginx/sites-available
# cp default default-orig
# medit default
```

The server block should match this example:

```
server {
listen 80 default_server;
listen [::]:80 default_server ipv6only=on;

# ssl support
listen 443 ssl;

root /usr/share/nginx/html;
index index.html index.htm;

# Make site accessible from http://localhost/
server_name localhost;
ssl_certificate /etc/nginx/ssl/nginx.crt;
ssl_certificate_key /etc/nginx/ssl/nginx.key;

location / {
# First attempt to serve request as file, then
# as directory, then fall back to a 404.
try_files $uri $uri/ =404;
# Uncomment to enable naxsi on this location
# include /etc/nginx/naxsi.rules
}
}
```

Relaunch nginx using the commands:

```
# sudo service nginx stop
# sudo service nginx start
```

4 - Lighttpd

Enter the certificate information as indicated for the nginx case. Similarly check installed certificate.

In addition, generate the corresponding .pem file using the commands:

```
# cd /etc/lighttpd/ssl
# cat lighttpd.key lighttpd.crt | sudo tee \
lighttpd.pem
```

This .pem file is later used in the lighttpd configuration file.

Add SSL configuration to the lighttpd configuration, using the following steps:

```
# cd /etc/lighttpd/
# cp lighttpd.conf lighttpd.conf-orig
# medit lighttpd.conf
```

Add the following block:

```
$SERVER["socket"] == ":443" {
ssl.engine = "enable"
ssl.pemfile = "/etc/lighttpd/ssl/lighttpd.pem"
server.document-root = "/var/www"
}
```

Check if the configuration update is ok, using the (single-line) command:

```
# sudo lighttpd -t -f \
/etc/lighttpd/lighttpd.conf
Syntax OK
```

Relaunch lighttpd using the commands:

```
# sudo service lighttpd stop
# sudo service lighttpd start
```

Relaunch firefox and point to the ip-address of the device, using the HTTPS URL `https://<u3-ip-address>`.

On the first access, certain dialog boxes will be prompted seeking a response.

Select the following options:

```
I understand the Risks
Add Exception
```

4 - Nginx

Relaunch Firefox and point to the ip-address of the device, using the HTTPS based URL `https://<u3-ip-address>`.

On the first access, certain dialog boxes will be prompted seeking a response. Select the following option buttons:

```
I understand the Risks
Add Exception
Confirm Security Option
```

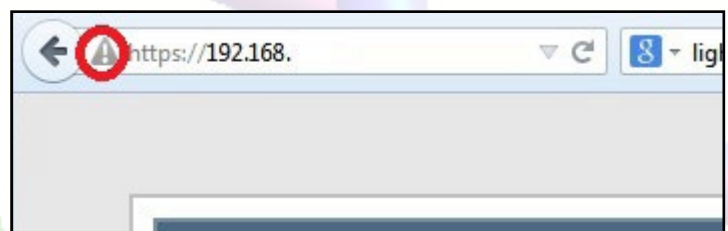
The welcome screen should appear as shown in the figure to the right.

The circled exclamation icon indicates SSL has been configured OK. Note that the exclamation indicates that a self-generated certificate is in use.

4 - Lighttpd

The welcome screen should appear as shown in the figure below.

The circled exclamation icon indicates SSL has been configured OK. Note that the exclamation indicates that a self-generated certificate is in use.



5 - Nginx and Lighttpd

Install PHP5-FPM and other modules using the command:

```
# apt-get install php5-cgi autoconf automake autotools-dev curl libapr1 libtool curl libcurl4-openssl-
dev php-pear php-xml-parser php5 php5-cli php5-common php5-curl php5-dev php5-gd php5-sqlite php5-fpm php5-
mysql
```

6 - Nginx

Configure nginx for php support, using the following configuration (socket-based) options:

```
# cd /etc/nginx/sites-available/
# medit default

...

index index.html index.htm index.php;

location ~ .php$ {
    fastcgi_split_path_info ^(.+\.php)(/.+)$;

    fastcgi_pass unix:/var/run/php5-fpm.sock;

    fastcgi_index index.php; include fastcgi_
params;
}
```

6 - Lighttpd

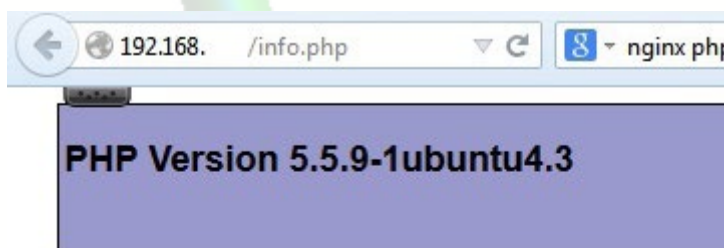
Update lighttpd for php support, using the following (socket-based) configuration options:

```
# cd /etc/lighttpd/conf-
available/
# cp 15-fastcgi-php.conf 15-fastcgi-php.conf-orig
# medit 15-fastcgi-php.conf
```

Add the following (socket-based) configuration:

```
fastcgi.server += ( ".php" =>
    (
        "socket" => "/var/run/php5-fpm.sock",
        "broken-scriptfilename" => "enable"
    )
)
```

Enable additional fastcgi configuration, using the commands:



PHP Version 5.5.9-1ubuntu4.3

System	Linux u3-2 3.8.13.26 #1 SMP PREEMPT
Build Date	Jul 7 2014 17:14:15
Server API	FPM/FastCGI
Virtual Directory Support	disabled
Configuration File	/etc/php5/fpm

Once you've reached this screen, make sure to congratulate yourself for becoming an expert level web server installer!

6 - Lighttpd

```
# lighttpd-enable-mod fastcgi
# lighttpd-enable-mod fastcgi-php
```

These commands create files/links:

```
# ls -lsa /etc/lighttpd/conf-enabled
total 0
lrwxrwxrwx 1 root root 33 Jul 13 16:14 10-fastcgi.conf -> ../conf-available/10-fastcgi.conf
lrwxrwxrwx 1 root root 37 Jul 13 16:15 15-fastcgi-php.conf -> ../conf-available/15-fastcgi-php.conf
```

7 - Nginx and Lighttpd

Update the fpm configuration:

```
# cd /etc/php5/fpm/pool.d/
# cp www.conf www.conf-orig
# medit www.conf
```

Add the following (socket-based) configuration:

```
listen = /var/run/php5-fpm.sock
```

Update the php configuration file:

```
# cd /etc/php5/fpm/
# medit php.ini
```

Enable the following config line:

```
cgi.fix_pathinfo=1
```

8 - Nginx

Create a sample .php script file, using the commands:

```
# grep -w '^[^#]*root' \
/etc/nginx/sites-available/default root \
/usr/share/nginx/www;
# cd /usr/share/nginx/www
# echo '<?php phpinfo(); ?>' > info.php
```

Relaunch nginx using the commands:

```
# service php5-fpm stop && sudo service nginx stop
# sudo service nginx start && service php5-fpm start
```

Relaunch Firefox and navigate to the .php file using the URL <http://<u3-ip-address>/info.php>

Verify that the PHP Info page is displaying properly.

8 - Lighttpd

Create a sample .php script file, using the commands:

```
# cd /var/www
# echo '<?php phpinfo(); ?>' > info.php
```

Relaunch lighttpd using the commands:

```
# service php5-fpm stop && \
sudo service lighttpd stop
# sudo service lighttpd start && \
service php5-fpm start
```

Relaunch Firefox and navigate to the .php file using the URL <http://<u3-ip-address>/info.php>

Verify that the PHP Info page is displaying properly.

Additional Resources

Read more about nginx at <http://wiki.nginx.org>, and learn about Lighttpd at <http://lighttpd.net>.

ODROID-VU AFFORDABLE 9" USB HDMI TOUCH SCREEN

A PORTABLE
MULTITOUCH SCREEN
FOR ANDROID, LINUX
AND WINDOWS

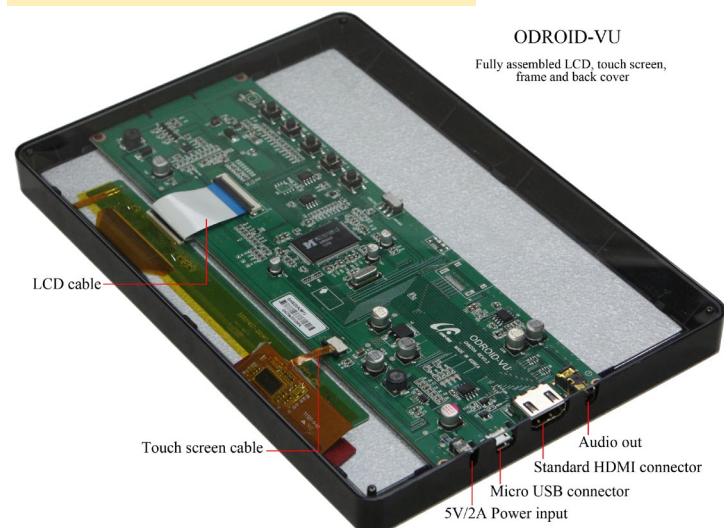
by Justin Lee

You'd think that, with the massive growth in tablet computing, it should be an easy matter to:

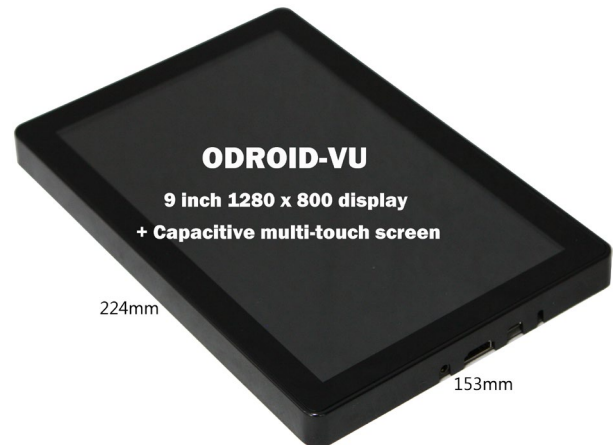
1. Locate a small high-definition (HD) LCD screen
2. Connect it to an HDMI/LCD driver board
3. Attach a capacitive multi touch screen on the LCD
4. Hook that up to your ODROID or other HDMI device

Unfortunately, it's not quite that simple at all, or we'd all have done it already! The ODROID computers can

A back view of the Odroid VU components



ODROID-VU
Fully assembled LCD, touch screen,
frame and back cover



The Odroid VU - A 9 Inch 1280 x 800 Multi Touch Screen Display

display 1920x1080, and you can buy a full-size monitor at that resolution, but that's not what we're about. We're looking for something small and portable, which is where it gets a bit tricky.

There are very few small, portable, HDMI screens available on the market. Most of the ones out there have 800x480 or 800x600 pixels. These are largely aimed at professional photographers and videographers, with

a price tag to match! And most of them have no touch screen option. So we decided to develop a 9-inch universal HDMI screen with 10-points capacitive multi touch input for ODROID users. The name of display is ODROID-VU, which is pronounced as "view". We hope this product can be a "View point" for your computer.

Specifications

Display: 9-inch 1280 x 800 pixels TFT LCD

Touch screen: Capacitive multi-touch 10 points input (USB HID)

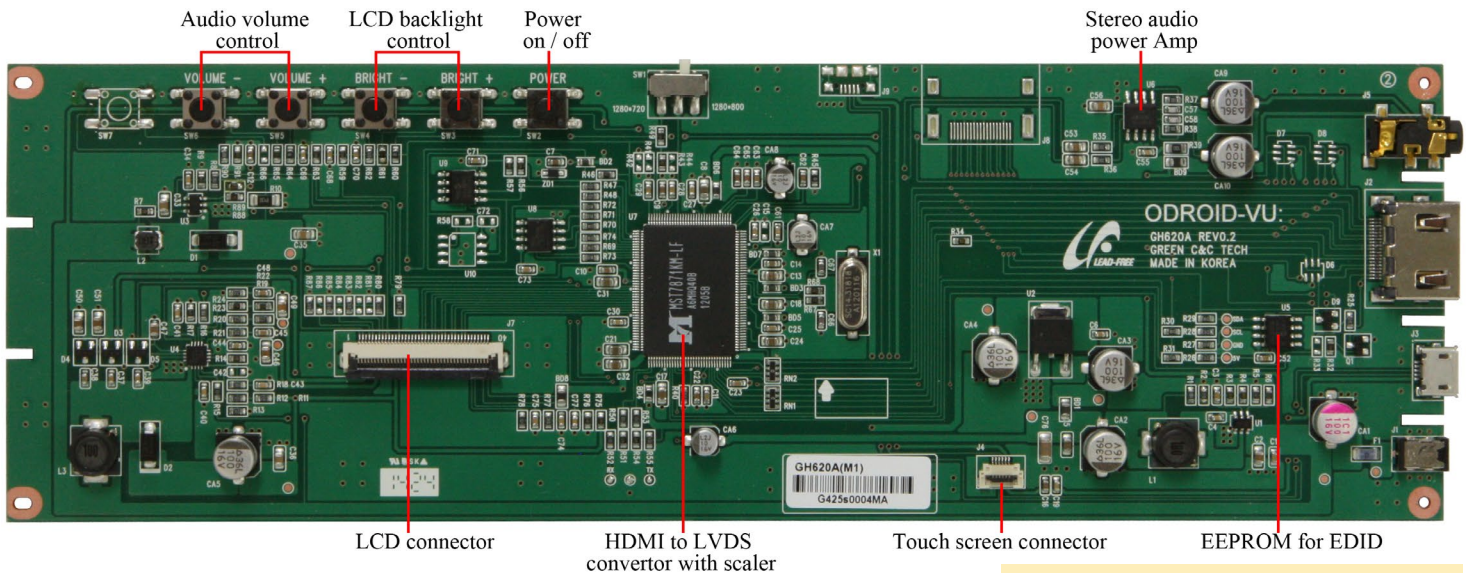
Dimensions: 224 x 153 x 11 mm with the plastic frame.

Power: 5V/1A DC

Ports: Power, Micro-USB, Type-A HDMI, Audio jack (3.5mm stereo)

5 keys: Volume control, Brightness control, Power on/off

The ODROID-VU is available from the **Hardkernel Store** at hardkernel.com

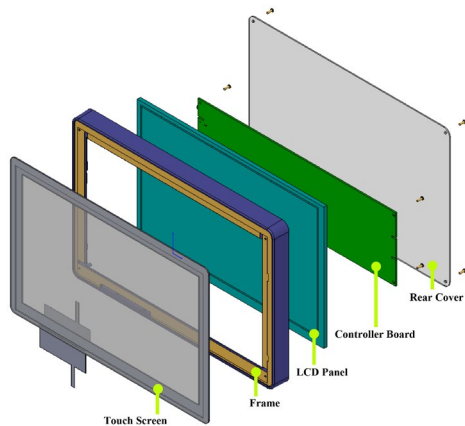


ODROID-VU board closeup detail

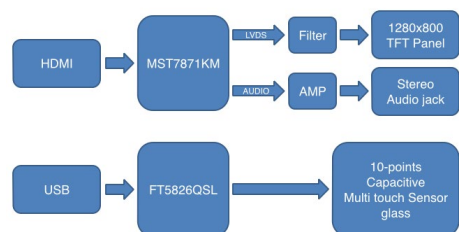
What's inside

The HDMI to LVDS converter IC MST7811KM has an HDMI receiver and an LVDS transmitter with analog audio output. The IC also has an internal frame buffer to scale up/down the image to fit into the 1280x800 output. Volume up/down and Backlight brightness control are also managed by the IC with On-Screen user interface.

The single-chip capacitive touch panel controller FT5826QSL supports mutual capacitive touch panel up to 10-



Block diagrams of ODROID-VU assembly



points at the same time. It also has the USB HID Multi-touch standard protocol which is compatible with Linux and Windows.

The USB VID:PID is 2808:81c9 which is very important information when you modify the HID kernel driver.

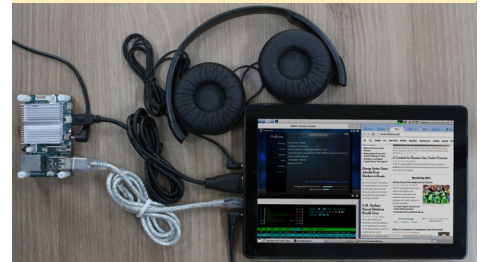
Use Cases

You can connect the ODROID-VU to ODROID boards as well as PC, TV and other embedded boards if they have HDMI output.

Some ODROID boards can't generate a proper HDMI master clock for 1280x800 native resolution of ODROID-VU. So 1280x720(720p) is scaled up to 1280x800 to fit the full screen. 1920x1080 input is also supported via scale-down function.

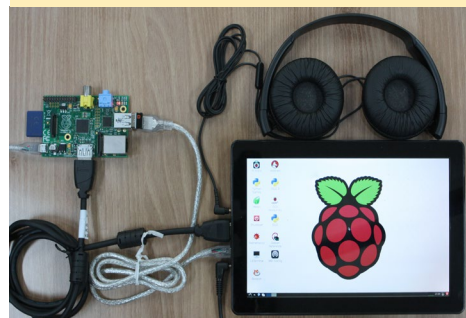
You can purchase the ODROID-VU, which comes with a power supply and micro-USB cable, at the Hardkernel store (<http://bit.ly/UmZEod>).

XBMC, Chromium and Terminal on Ubuntu with U3 on the ODROID-VU



ODROID-U3 running Android 4.4 with an ODROID-VU attached as the main screen

Raspberry Pi and Debian with ODROID-VU



Laptop running Windows 8.1 with an ODROID-VU used as a side monitor



PEPPERFLASH CHROME PLUGIN FOR LUBUNTU 14.04

AN EASY WAY TO WATCH
ADOBE FLASH ON YOUR
ODROID LINUX SYSTEM

by @Miltos

edited by Venkat Bommakanti



The Adobe Flash Player for the Chrome web browser in Ubuntu was discontinued by Adobe several years ago, with no alternative available for playing Flash videos in Linux until recently. Google recently released an updated Flash-compatible player plugin called PepperFlash, which includes an ARM Ubuntu version. This article describes the steps needed to download this plugin from Google and add it to the Chrome web browser running Lubuntu 14.04 LTS on any ODROID board, including the X, U and XU series.

Requirements

1. Any ODROID board, with an appropriate power adapter.
2. A bootable 8+ GB MicroSD card or eMMC module containing the latest Lubuntu image available from the Hardkernel website.
3. A network where the device has access to the Internet and the ODROID forums.
4. Optional SSH access to the ODROID via utilities like PuTTY (MS Windows 7+) or Terminal (Mac, linux to perform the steps from a remote host computer.

5. PepperFlash Ver. 12.0.0.77 for Lubuntu 14.04
6. PCMan File Manager (pcmanfm) for Lubuntu 14.04

Download PepperFlash

Login to the ODROID using the default username of “odroid”, and download the precompiled packaged version of the ODROID compatible PepperFlash by navigating to <http://bit.ly/1yYEDQf> in the Chrome browser window, which will be saved to the /home/odroid/Downloads directory.

Launch a terminal session and navigate to the downloads directory, then launch pcmanfm with root privileges by typing the following command in

the Terminal window:

```
$ sudo pcmanfm
```

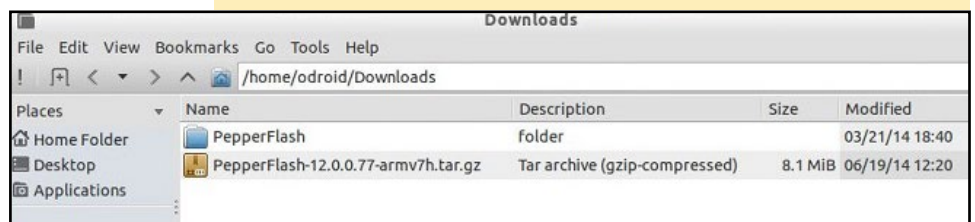
Right-click the icon for the downloaded tarball, which should be visible in the right-most pane, and selecting the Extract Here menu option.

The plugin will be extracted to the /home/odroid/Downloads/PepperFlash directory, as shown in the second screenshot.

It is easier for some users to use the command-line terminal window almost entirely for these next steps. However, using pcmanfm simplifies the process for those who want to save some typing.

In pcmanfm, copy the PepperFlash directory and paste it into the /usr/lib directory.

Contents of the directory after the PepperFlash tarball is extracted



Configure Pepper Flash plugin

After closing `pcmanfm`, launch the `medit` (installed by default in Ubuntu) editor and open the default configuration file of PepperFlash by using the command:

```
$ sudo medit \
/etc/chromium-browser/default
```

Modify the `CHROMIUM_FLAGS` configuration parameter as a single line:

```
CHROMIUM_FLAGS=" --ppapi-flash-
path=/usr/lib/PepperFlash/ \
libpepflashplayer.so --ppapi-flash-
version=11.7.700.225"
```

Save the configuration file and close `medit` as well as Chrome.

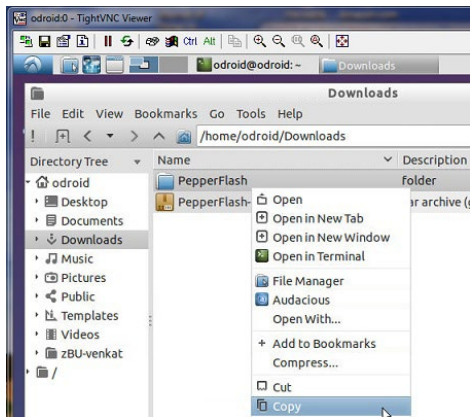
Validate the installation

Verify that the PepperFlash plugin has been installed correctly by re-launching the Chrome web browser and navigating to the URL `chrome://plugins`. Select the PepperFlash plugin and click on the Details option at the right side of the webpage.

Check the details ensuring the plugin information matches what is listed in the screenshot. Finally, enable the Always allowed option by checking the checkbox and close all Chrome windows.

To make sure that the plugin is installed properly, reboot the ODROID, then launch Chrome again. Navigate to any page with Flash content, such as the Adobe Flash Samples website at <http://adobe.ly/UmTzrV>.

For additional information or questions, please visit the original information sources at <http://bit.ly/1lstuyP>.



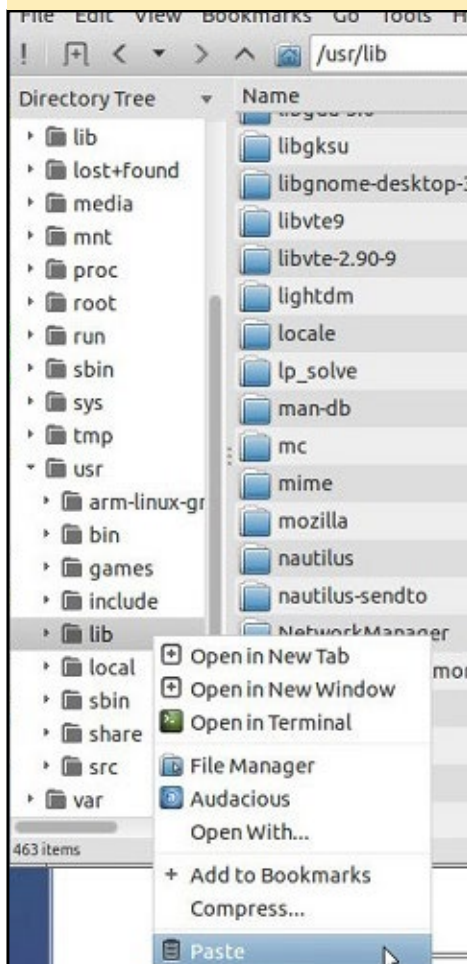
Copy the PepperFlash directory

After pasting the copied directory into the system library directory at `/usr/lib/`, the directory `/usr/lib/PepperFlash` will be created.

If using the command line instead, copy the PepperFlash directory to the system library directory by typing the following command:

```
$ sudo cp PepperFlash /usr/lib/
```

Paste the PepperFlash directory



MUPEN64PLUS TURN YOUR ODROID INTO A NINTENDO 64 RETRO GAMING CONSOLE

by Rob Roy



Mupen64Plus is a Nintendo 64 emulator for the Android platform, and it runs great on the ODROID!

To install Mupen64plus, visit the Mupen64Plus Play Store page at <http://bit.ly/YzRD1H>, connect your joysticks, and immerse yourself in the glory that was the Nintendo 64!

From top to bottom: F-Zero-X, Mario World 64, and Mario Tennis 64



IO SHIELD DEMYSTIFIED

HOW TO CREATE AN INTERMEDIARY BETWEEN THE HARDWARE AND THE HUMAN

By Bohdan Lechnowsky

I was recently contracted to work on a project that would require a U3 to potentially control valves, actuators and other devices and read inputs. That was exciting, because I knew I'd have a chance to work with the U3 IO Shield -- something that I've wanted to do since it was released, but haven't had a chance to do because of my busy work schedule.

My first step when working with any new hardware is to see if I can get it to work using the supplied examples, and my second step is usually to post questions to forum.odroid.com about setup issues I discover. This continues until I succeed in getting it to work. The third and fourth steps, however, are ones that I particularly enjoy.

The challenge of the third step is to boil down the functionality of the new hardware to the most simple, elegant, flexible and efficient form as possible while throwing out any preconceived notions of how that hardware is currently designed to be accessed or controlled. This doesn't involve any coding, but rather is a process of designing a dialect to act as an intermediary between the user and the hardware while achieving all the stated goals.

The fourth step is to convert that dialect into code using Rebol's powerful parse feature. Amazingly, once the hard part of the third step is done, the fourth part is quite simple. Most often

during the fourth step, I find it necessary to fine-tune the design produced during the third step. Of course, the fourth step also involves testing and refining of the design based on the results of testing.

Steps 1 and 2

When I first attempted to use the IO Shield on Ubuntu, I had trouble finding concise instructions for getting it to work using the method that is easiest for Rebol to interface with. That took me to the forums where I was eventually able to piece together this set of basic instructions:

Update the Ubuntu kernel by following instructions in the forum post at <http://bit.ly/1o3vMJx>.

Type the following into Terminal:

```
odroid@odroid:/sys/class/gpio$
sudo su
root@odroid:/sys/class/gpio# modprobe gpio-pca953x
root@odroid:/sys/class/gpio#
modprobe i2c-gpio-custom
bus0=4,200,199
root@odroid:/sys/class/gpio# echo
tca6416 0x20 > /sys/devices/platform/i2c-gpio.4/i2c-4/new_device
root@odroid:/sys/class/gpio# echo
304 > export ;304 is equivalent
to P17 on the IO Shield
```



The U3 I/O Shield is a powerful tool in an ODROID wizard's magic kit

```
root@odroid:/sys/class/gpio# cd
gpio304
root@odroid:/sys/class/gpio/
gpio304# echo out > direction
root@odroid:/sys/class/gpio/
gpio304# echo 1 > value ;Turns on
the pin
root@odroid:/sys/class/gpio/
gpio304# echo 0 > value ;Turns
off the pin
```

This is what I needed to start developing my IO-Shield dialect, which I decided to name gpio.

Step 3

After quite a bit of thought and design, here are some examples of what I wanted my dialect to be able to do:

```
gpio [
init out P17 ;initialize pin
P17 as an output
on ;turn on P17
wait .1 ;wait .1 seconds
off ;turn off P17
deinit P17 ;deinitialize pin
P17
]
```

```

gpio [
  init in [P00 P01]    ;Set
pins P00 and P01 as inputs
  init out [P02 P03 P06];Set
pins P02, P03 and P06 as outputs
  init pwm 1 .1 P04    ;Set pin
P04 as a pulse-width-modulated
output with a
                    ; period of 1
second and a pulse of .1 seconds
  [P02 P06] on      ;Turn
pins P02 and P06 on
  init .1 .01 P05    ;Set pin
P05 as a pwm output - the last
initialization
                    ; (P04) was a
pwm, so it was remembered for
this init.
  init gpio in P07    ;To
change back to standard I/O,
specify gpio.
  wait .01           ;Wait .01
seconds
  P06 off           ;Turn pin P06
off
  P03 on            ;Turn pin
P03 on
  read speed P00     ;Read the
value of pin P00 into system/
gpio/val/speed
  read rpm P01       ;Read the
value of pin P01 into system/
gpio/val/rpm
  wait 5            ;Wait 5
seconds
  deinit [P04 P05]   ];Deini-
tialize pins P04 & P05 (this is
the way to turn off
; a PWM pin)
  wait 1            ;wait 1
second
  reset             ;Deinitial-
ize all standard I/O and PWM pins
]

```

Step 4

In order to get to this level of abstraction, the following hurdles had to be overcome by the dialect:

- Whenever gpio is called, determine whether the IO Shield subsystem had already been initialized. If it hadn't, initialize it.
- Convert the disparate timing systems to a standardized time system (PWM uses microseconds, wait uses seconds). Seconds are more easily handled by most people, so decided to standardize on seconds.
- Provide a level of abstraction between the printed port numbers on the IO Shield with the /sys/class/gpio port numbers.
- Allow handling standard I/O and PWM within the same dialect without any need for understanding the underlying mechanism.
- Reduce the required wordiness of the dialect by allowing single or multiple ports to be operated on simultaneously, and unnecessary words to not be required, but allowed when the user uses them, and to remember the last pin initialization types and ports so the following commands don't need to re-specify them unnecessarily.
- Testing had to be performed with different methods of reading/writing to/from the /sys/class/gpio and writing to the sys/class/soft_pwm directories for reliability and performance under different conditions.

Fortunately, incorporating this type of functionality within Rebol's parse is quite straightforward and easy once you become familiar with it.

Incidentally, parse uses a dialect to allow you to easily define the rules for your dialect! Because of this, if you can think

of an easier way to write a dialect, the open-source developers of Rebol would love to hear about it.

Testing

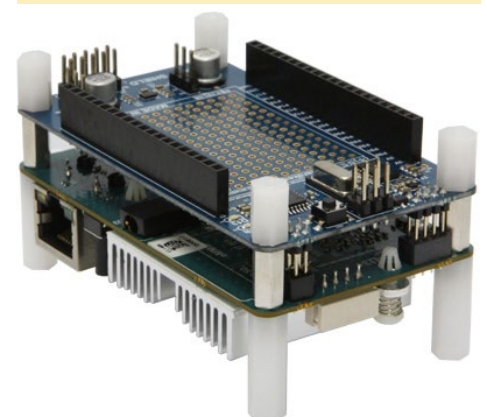
During testing on the U3, I found that utilizing PWM sometimes leads to instability in the operating system. I think this might have something to do with the constant interrupts required to make PWM work at high speeds in a software system. I also found that sometimes even at slow period/pulse speeds, PWM was sometimes laggy and inaccurate.

Also found during testing was that using Rebol's write command would sometimes not work, but other times it would work fine. I haven't yet been able to determine the circumstances that lead to this undesirable behavior. Rebol also has an echo command that would work sometimes even when write would not. The most reliable (but by far the slowest) method was to initiate a bash-based echo command from within Rebol. In my testing, this method was over 25 times slower than the native write and echo commands in Rebol.

Results

You can download the dialect for your own use at <http://respectech.com/odroid/gpio.r3>, where you can leave feedback. The gpio.r3 dialect definition script is editable, so feel free to make changes and improvements!

The I/O Shield is now yours to command!



DIGGING (INTO) THE ODROID-SHOW

UNLOCK THE ARDUINO HARDWARE'S FULL POTENTIAL

by Declan Malone

Previous issues of ODROID Magazine covered the basics of setting up the ODROID-SHOW hardware and using it as a display for various data generated by your ODROID or PC. While this has many useful applications, it only scratches the surface of what's possible with the SHOW. At the heart of the device is an ATmega328p chip, which is the same kind of microcontroller that's contained in many models of the popular Arduino platform. This means that almost anything that you could imagine using an Arduino for could just as easily be made to run on the ODROID-SHOW.

This series of articles aims to dig below the surface, so to speak, and to introduce you to using the ODROID-SHOW as a programming platform in its own right. In this installment, I'll cover the software components that come pre-installed on the ODROID-SHOW, how they work, and how you can modify them and upload your changes to the device. I'll use a simple example of a Mandelbrot Set viewer to demonstrate this.

In the second article, I'll deal with some options for connecting the SHOW to some simple electronic components via the two GPIO headers. Finally, the third article will look at building a complete, self-contained game on the hardware itself.

The latest version of the ODROID-



The ODROID-SHOW can do much more than display simple text data

SHOW example software can be downloaded from Hardkernel's official GitHub repository:

```
$ mkdir ~/src; cd ~/src
$ git clone https://github.com\
hardkernel/ODROID-SHOW
$ cd ODROID-SHOW
```

You will find directories named “example/”, where you will find programs to run on the ODROID or PC host, “show_main/”, where you will find the main program to run on the SHOW, and “libraries/”, where the various Arduino support libraries live. Depending on when you downloaded the sources from github, you may see other directories, too, such as one for the weather module, which I won't cover in these articles.

You may already have looked at the programs in the “example/” directory. If not, now might be a good time to have a look, so that you can familiarize yourself with some of the capabilities of the ODROID-SHOW. You can also check out the previous articles for more

information.

For this article, I'll be concerned with looking at contents of the “show_main/” and “libraries/” directories, which are explained in the block diagram shown on this page.

Parts explained

Terminals (meaning some kind of keyboard and display combination) have been around for over a century, even before the first computers were built! As computers became more commonplace, many of the old terminal devices were put to use as dumb data entry and display devices. In the early days, instead of personal PCs, it was much more common to have a large and powerful computer (such as a mainframe computer) connected to lots of different terminals.

Because of the wide variety of different kinds of terminal devices, as well as the wide range of features that newer, more sophisticated terminals offered (such as coloured text, an audible bell, ability to position the cursor or clear the screen, etc.), some sort of standardisa-

tion was needed. The list of “ANSI escape code” (such as “ESC[2J;” to clear the screen) was the result.

While this might sound like quite an archaic system (after all, we all have PCs, laptops and tablets now, not terminals!), actually it’s still in common use today. If you use “xterm” or a more modern equivalent (like “gnome-terminal”) you can still use the ANSI escape codes for all the things that used to be done on dedicated terminals.

More importantly for our discussion here, this is exactly what the “show_main” program block in the diagram above is implementing by programming the Arduino to behave as if it was a terminal screen.

You can check out the full list of “escape codes” that the software implements by referring to <http://bit.ly/1nkYKUt>. I’ll come back to this later when I add an ANSI-like escape for drawing a single pixel on the screen.

The other box on the left is for the serial connection to the PC (or ODROID) host. It’s used both for receiving data from the PC and also for printing debug information, which can be very handy when writing Arduino/SHOW programs.

On the right-hand side, we have two libraries that were also developed by Adafruit. The top one implements high-level graphical primitives, such as printing a character, drawing points, lines and circles, and filling in rectangular areas with solid colours. This can be seen as the platform-independent part of the stack.

Below that, we have the platform-specific part of the graphics code. It knows about how the ATmega chip is wired up to the TFT module, as well as the protocols needed to communicate with the actual hardware, as shown in the box below.

We won’t be delving too deeply into how this side works, but you may be interested in examining the code if that interests you.

Arduino Programming Basics

All the software components included in the official software for the SHOW device are written in C++. If you know C++ (or even C) already, you should have no problem starting out on programming for this platform. There are some minor some differences in the way C++ is used on the Arduino, and you should be aware of them. For example:

- Instead of using low-level commands like “gcc” and “make”, all your editing, compilation and uploading of programs is done in a graphical IDE (Integrated Development Environment) named, appropriately, Arduino.
- Programs written for the Arduino are often called “sketches”, and generally have a “.ino” file extension
- Programs do not have a “main()” function, since this is supplied by the Arduino compile system
- All initialisation is done in the program’s “setup()” function
- Arduino programs also require a “loop()” function. As the name suggests, this function is called in an infinite loop directly after the “setup()” function returns

These are all relatively minor differences to working on a “real” computer, so it’s actually quite easy to dive into developing programs for the Arduino (and SHOW!) platform.

Development Environment

For our first step with programming the SHOW, we’ll just compile and reinstall

the official software. Let’s get started by installing the Arduino IDE. On Debian/Ubuntu systems this is done by typing:

```
$ sudo apt-get install arduino
```

This will install a lot of other programs besides the IDE, including a cross-compiler (“avr-gcc”) and some supporting tools (“avrdude” being the main one). For the most part, you don’t need to worry about all these extra tools, since the Arduino IDE will call them transparently for you when it needs to. Later on, if you want to change over to using these tools directly (with your favourite editor and Makefiles), you’ll be able to do so. For these articles, however, I’ll stick with the simple IDE option.

There are four other important setup steps:

1. Tell the IDE where you will store your “sketches” (not where you downloaded the sources from github!)
2. Tell it which serial port to use to communicate with ODROID-SHOW (this may change if you reboot or move USB devices around)
3. Tell it which version of the Arduino is being used
4. Tell it where to find your libraries

All steps except Step 3 are covered in the official ODROID wiki page at <http://bit.ly/1p8uitU>, so I won’t repeat them here. For Step 3, select “Tools > Board > Arduino Uno”. The ODROID-SHOW is fully compatible with the pre-built Arduino Uno configuration.

If you are running an older version of the Arduino program (for example, my Debian Wheezy distribution only provides version 1.0.1 of the program), then in Step 4 there may not be an “Add Library ...” option under the “Sketch > Import Library” menu option. If your

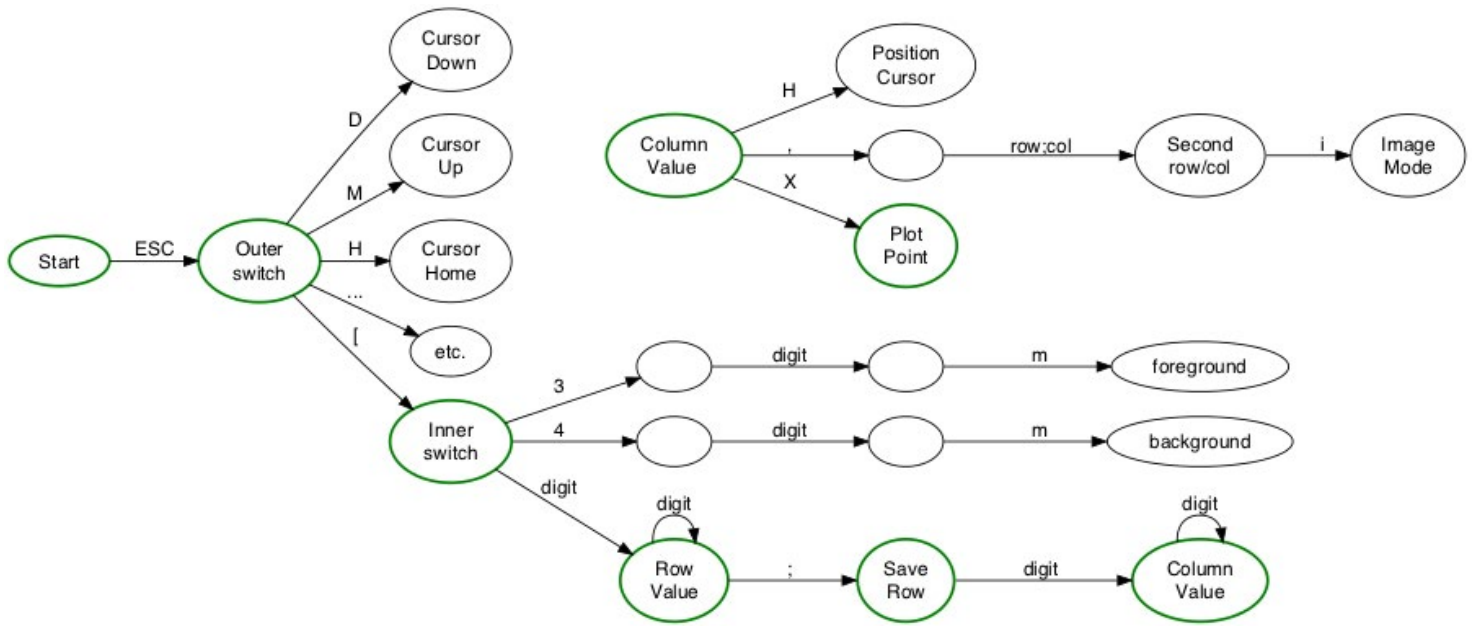


Diagram of the Finite State Machine for the ANSI parser

Adding a “plot” command

version of Arduino is too old like mine, then you will have to add the “GFX” and “ILI9340” libraries manually by creating symbolic links from the “libraries” directory of your sketchbook directory (set up in Step 1) to the individual library directories as contained in the code that you downloaded from Github. For example, if you downloaded into the “~/src/ODROID-SHOW” directory, you can create the links with:

```
$ cd ~/sketchbook
$ mkdir libraries; cd libraries
$ for d in \
  ~/src/ODROID-SHOW/libraries/*;
do ln -s $d .; \
$ done
```

After these setup steps, we can load “show_main.ino” into the IDE. After that, there’s only one more thing to do before reprogramming the device, and that’s to attach the programming jumper. This is used to short-circuit a set of two pins labelled P2 on the board. They’re located between the reset switch (marked ‘Reset’) and the power LED (marked ‘Alive’). Once you’ve done that, all you have to do is click on

the green “right arrow” mark in the row of icons directly underneath the menu bar.

If all goes well, you should get some sort of indication in the message panel at the bottom, and the SHOW will then reboot. After a successful flash and reboot, you’ll need to remove the jumper before running any tests, such as those in the “examples/” directory. Jumpers tend to be small and fiddly, so try to remember where you put it after removing it! Alternatively, leave it attached to just one of the pins on the P2 header. You can also get different types of jumper that have an extra tab on top. They tend to be a lot easier to attach and remove and are a little bit harder to lose because they’re bigger. You don’t need to have one of these, but I find that it helps.

If things didn’t go well, then you can check the message panel to give some idea of what went wrong. You’ll get different messages depending on whether the compilation failed (eg, due to missing libraries or syntax errors) or there was a problem with uploading (like if you forgot to reattach the jumper). It’s pretty user-friendly and you should be able to figure out what went wrong based on these messages.

When I first used the ODROID-SHOW (v1.1 of the software), it supported lots of ANSI escape characters for drawing text on the screen, but it didn’t include any other graphics primitives. I decided to change that by adding a new escape sequence to let it plot a single point on the screen. Here’s how I went about adding that feature.

The first thing that I did was to see if there was an existing ANSI escape code for plotting points. As it turns out, there isn’t, so I chose this as the syntax for the new command:

```
ESC [ row ; col X
```

By examining the code in “show_main.ino”, You can see that the “parsechar()” function is called every time a new character is read from the PC over the serial port. The parser uses what’s called a “Finite State Machine” (or “FSM”). This is quite commonly used in any application where you have to build up complex commands one character at a time.

A FSM-based parser is simply a list of “states” (which track partially recognised commands) and “transitions” (which are followed by seeing a new character from

the input). Please refer to the diagram to the right, which shows some of the FSM for the ANSI parser.

This is not a 100% accurate or complete depiction, but it should help to explain how the parser works in general. I have highlighted in green the path that would be taken for the parser to recognise my new point-plotting command. As you can see, the code is simply “grafted” onto the existing graph structure. Once I knew where the new code would go, the code to implement the new command was very simple:

```
case 'X':

row = (row > right_edge0) ?
right_edge0 : row;

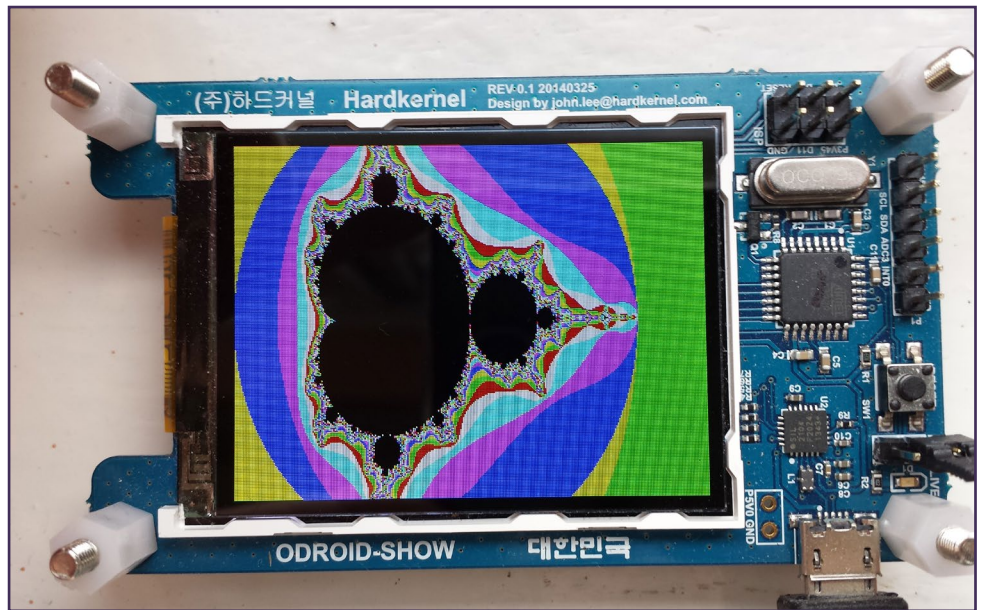
col = (tmpnum > bottom_edge0) ?
bottom_edge0 : tmpnum;

tft.drawPixel(row, col,
foregroundColor);

break;
```

This does bounds-checking on the supplied row and column values then calls the appropriate method from the GFX object (“tft.drawPixel()”) using the current foreground colour.

These changes were since incorporated into a later version of the official



ODROID-SHOW displaying a Mandelbrot set

software, so you can check for yourself where the code was added.

If you can understand how the parser is organised, then it should be possible to add more features to it by following the same method as above. You could, for example, add more escape sequences for drawing circles, lines or boxes, all of which are available from the GFX library but not currently exposed to the PC side.

It would be quite straightforward to implement a set of LOGO-style primitives, for example, with commands for moving forward or backward, rotating the “turtle”, putting the pen up or down, and so on.

Mandelbrot Set

A common first step when checking out any new graphics hardware is to display the Mandelbrot Set on it. In the unlikely event that you’ve never heard of this before, the Wikipedia page at <http://bit.ly/1mQ75y3> is a good place to start. That page also includes pseudo-code for generating the Set, so I won’t need to cover that here.

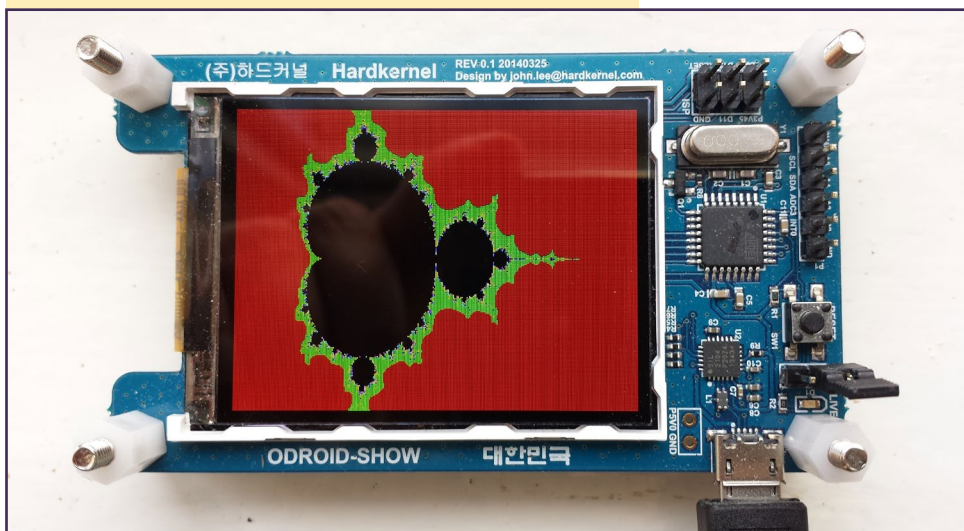
You can download my code from a fork I made of the official Hardkernel SHOW repository by typing the following commands:

```
$ cd ~/src

$ git clone \
https://github.com/declanmalone/
ODROID-SHOW.git SHOW-fork

$ cd SHOW-fork
```

ODROID-SHOW displaying a Mandelbrot set



There are actually two versions of the code included here. The first, in the “examples/” directory is called “mandelbrot.c”, and it’s intended to be run on the host (PC) side. A “Makefile” is included to compile it automatically on an ODROID machine or another Linux system. There are a few things to note about this code:

1. It includes some code based on the official “port_open.c” program. This code opens the USB serial port and sets the baud rate and other important parameters. This means that the code does not need to have a separate “port_open” program running.
2. The “plot_point()” function sends the appropriate “ESC [x ; y X” string over the serial port. It also sends a separate “ESC 3 ? m” string to set the colour of the point.
3. That function also includes appropriate error-checking to make sure that the full string was actually written to the device. If the send buffer overflows, then “write()” will fail and set the error value to “EAGAIN”. The code correctly handles this case by advancing the write buffer and trying the “write()” again until the full string has been sent.
4. There is a call to “usleep()” after every write. This is needed because the ATmega is only a single-threaded computer and if the PC side sends data too quickly, it is possible that some of the data will be lost. Adding a short, user-adjustable delay after every write is a simple way of working around this.

Although I wrote this version of the program to use the “X” escape code, it would also be possible to send the data across by first sending an “ESC [x1 ; y1 , x2 ; y2 i” escape sequence and then sending the colour data for each pixel as if you were sending an image.

Please refer to the “images.sh” example program from the official sources to see how that can be accomplished.

Native Generator

I also wrote a version of the Mandelbrot Set program as a sketch designed to run independently on the ODROID-SHOW, without needing any host PC. The sketch can be found in the “duino_fractal/” directory. Compiling and installing is done in the same way as you did for the official terminal emulator software, as described earlier.

The native code is broadly similar to the version for the PC, with some differences:

- It’s written in Arduino-style C++, with a “setup()” and (an empty) “loop()” function
- It makes use of the GFX and ILI9340 libraries directly
- It sends back debug messages over the serial connection with “Serial.print()” calls
- It also uses the Arduino’s internal timer function “getMillis()” to time how long it takes to render the screen
- It writes to the TFT screen’s memory directly rather than calling “drawPixel()”. This means that it calls “setAddrWindow()” once for each “panel”, then write each pixel’s colour

data with two calls to “spi-write()”.

It takes the ATmega chip just over two minutes to render the full image with a maximum iteration value of 1000. Considering that the chip has no floating point capability and it only runs at 16MHz, this isn’t too bad. The native code also has one advantage over the PC code: since it doesn’t receive any data over the serial connection, it doesn’t need to delay after writing each pixel.

When I wrote the code, I had initially intended to use a potentiometer and two push buttons to allow the user to zoom in and out on a particular screen area (in a similar way to the video at <http://bit.ly/1yZ0UgB>), but I never completed that part. As a result, only some of the code to support this, such as definition of hardware pins and ability to paint individual “panels”, for easier scrolling, are actually implemented.

Summary

We’ve had a lot of ground to cover in this article. I hope that it has served as a good introduction to the basics of programming the ODROID-SHOW and has helped to explain some of the things that the platform is capable of. I think you’ll agree that it’s actually not that difficult to get started with writing code for it. In the next article, I’ll get to grips with hooking up the SHOW to other components via the GPIO pins.



OS SPOTLIGHT: POCKET ROCKET AND COUCH POTATO

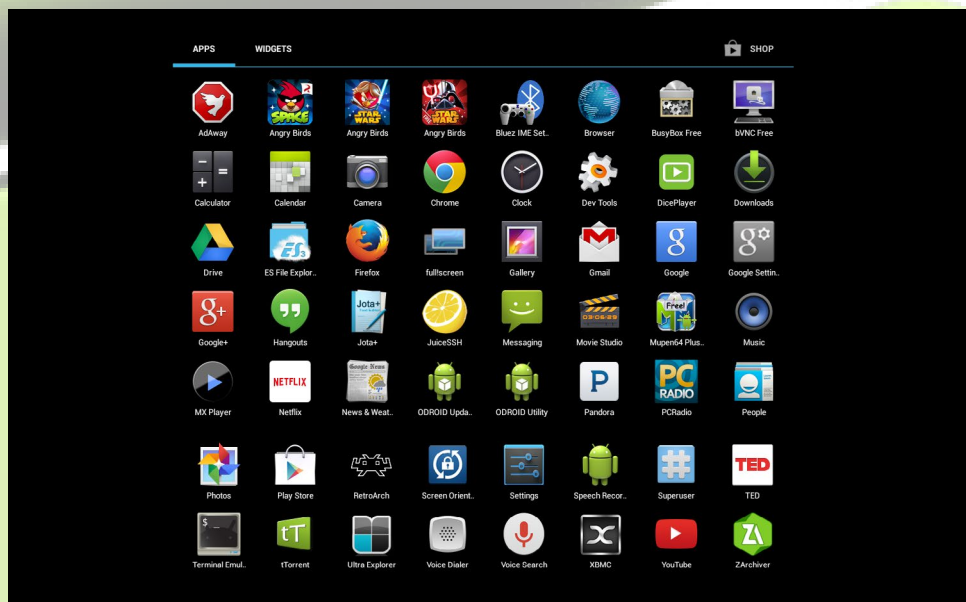
ANDROID 4.X PREBUILT IMAGES
FOR THE ULTIMATE SET-TOP BOX
NOW AVAILABLE IN KITKAT

by Rob Roy, Editor-In-Chief

The ODROID family of computers was originally marketed as an Open Android system, offering application programmers an alternative to purchasing an expensive Android phone and rooting it. ODROIDs runs Android extremely well, while including many useful features such as hardware video decoding and joystick drivers, which would need to be installed separately on a comparable Linux X11-based system.

When I began using ODROIDs, I set up a set-top box using an Ubuntu desktop environment called LXDE, which let me stream and download media files, then play them using Xine or XBMC. It could render 720p videos well, but only some 1080p videos, with lots of dropped frames, and was watchable but choppy. After trying several tweaks to achieve smooth video, including overclocking and setting kernel options, I reformatted my SD card and installed Android Jelly Bean 4.1.2 on my U2 instead.

The Jelly Bean 1080p video experience was incredible, and exactly what I was looking for! Using both XBMC and MX player, the ODROID could smoothly play even the densest videos on the XBMC website (such as Big Buck Bunny) that are known to bring other systems to a stuttering halt. The video decoding on every Android build (X,



The list of pre-installed apps for Pocket Rocket and Couch Potato

U, and XU) is flawless when using MX Player, and plays videos of any resolution and encoding.

At the time, I had also purchased an Apple TV, which has a friendly interface, but I couldn't install new applications, since the iTunes store isn't available for the Apple TV. However, an ODROID Android system allows unrestricted installation and removal of applications just like an Android phone, with thousands of applications and games available from the Google Play Store. Android is already rooted, so the operating system can be customized, self-writte apps can be installed, and administrative functions can be performed as the root user.

Before purchasing my first ODROID, I had never before considered using Android as my main operating system, but after putting together the Pocket Rocket image, I learned the advantages of it. Android is an extremely stable OS, and even when an app crashes or starts to slow down, Android makes it easy to shutdown the application. It also has a shallow learning curve, with large, friendly icons, as well as shortcut buttons that are intentionally similar to other operating systems.

In order to use Android efficiently, I use the built-in keyboard shortcuts, which are similar to other desktops such as Windows, Ubuntu and OSX.

Pocket Rocket and Couch Potato for the U, X and XU series may be downloaded from <http://bit.ly/lzYsZFt>.

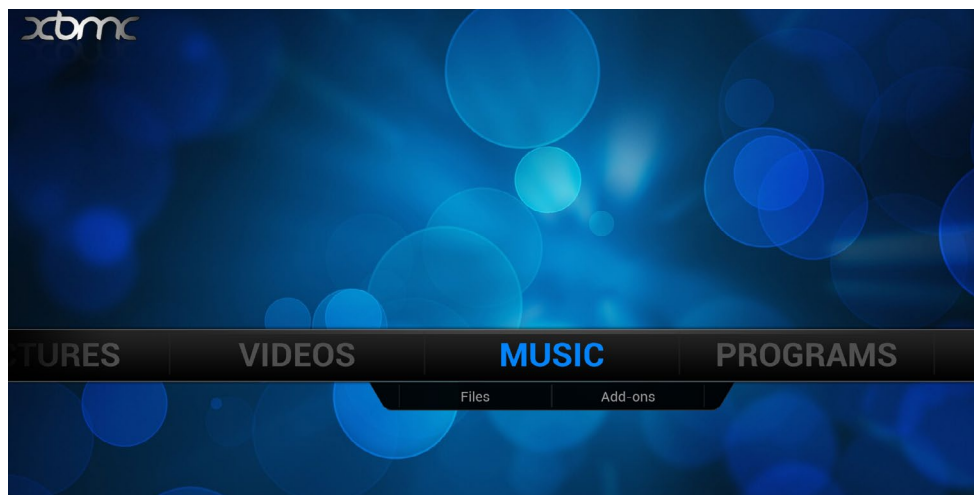
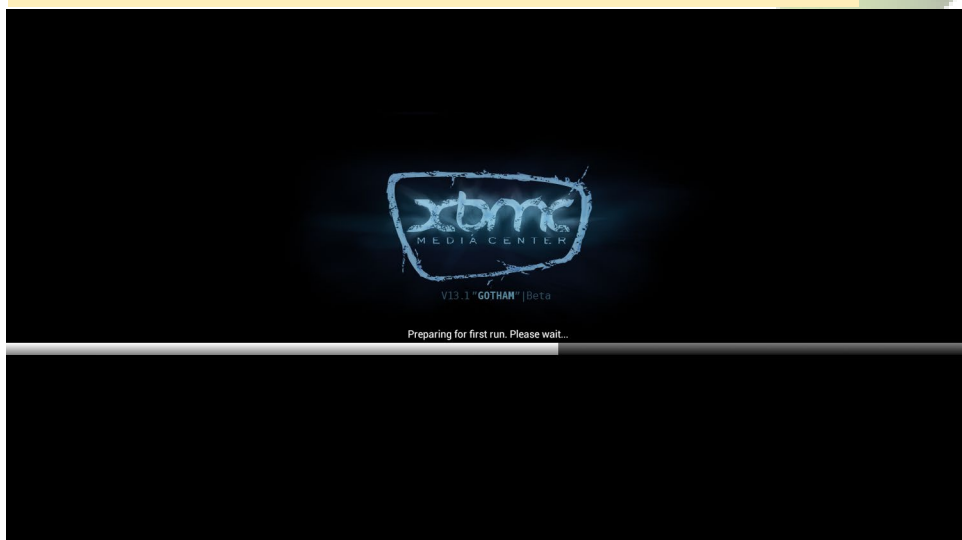
Keyboard Shortcuts

Ctrl-C	Copy
Ctrl-V	Paste
Ctrl-A	Select all
Ctrl-W	Close Window
Ctrl-Right Arrow	Move right one word
Ctrl-Left Arrow	Move left one word
Ctrl-Shift-Right Arrow	Select word to the right
Ctrl-Shift-Left Arrow	Select word to the left
Alt-Tab	Switch program
Alt-Esc	Desktop
Control-L	Go to address bar

Android Experience

Android is especially suitable for beginners, or anyone that has experience with a mobile phone interface. However, even as an expert computer user, sometimes I just want to change something without having to adjust settings, navigate menus or move windows. Android is perfect for this, and can perform

XBMC Gotham 13 starting up for the first time. This is the final release of XBMC, since they are changing the name to Kodi for version 14.



XBMC is the go-to program for all of your media center needs. It supports video, music, addons, and even can launch other applications.

basically any task that a desktop PC can perform, but in a different way.

As an example, most desktop PCs offer a word processor, and save its files to the local hard drive. In Android, the software used for editing documents is called Google Drive, which stores the information on the cloud. With Google Drive, you can pick up any mobile device, laptop, tablet, or desktop PC, log into Google Drive, and have access to all of your documents without needing to carry a USB thumb drive with you.

One notable difference with Android is that the right mouse button is actually used as the “Back” button. To bring up menus that would normally be done with a right click, use a “long click” in-

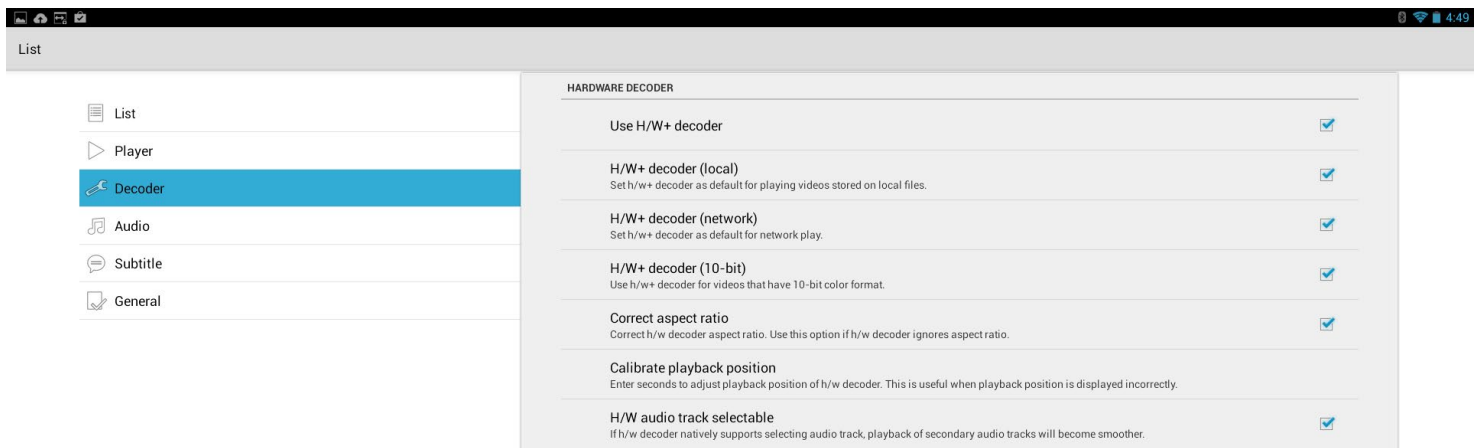
stead by depressing the left mouse button for several seconds. For example, in ES File Explorer, to select multiple folders, long click the first folder, which will then bring up a checkbox interface, where other folders can be selected with a single click.

eMMC vs SD card

Android is designed to run from an eMMC module, and a distinct advantage of ODROID computers is that this module can be removed and replaced easily. Although an SD card will technically run Android, the slow speed of the SD card hinders the experience because of the disk I/O. I stopped releasing Pocket Rocket images on SD card because of the amount of time spent clicking the Android “not responding” popups while customizing the image.

However, I discovered that the Ice Cream Sandwich (ICS) version of Android runs smoother than Jelly Bean, when comparing them both using an SD card. As a result, I created a special SD card version of Pocket Rocket called Couch Potato that has all of the same applications as Pocket Rocket, but uses ICS for better disk performance.

If you don't have the eMMC module, then you can install Couch Potato instead, and enjoy all of the same features as Pocket Rocket. Occasion-



MX Player is the best media player available for Android on the ODROID, and offers smooth hardware video decoding in 1080p.

ally, Couch Potato will interrupt with a “not responding” popup, such as when downloading multiple applications from the Play Store, but they are less frequent than when running Jelly Bean from SD card. Both images offer the exact same applications, and only differ in the version of Android 4.0 vs. 4.1.

Google Play Store

The Google Play Store is accessed by pressing the Apps icon made of 6 small squares on the upper right of the Android desktop, then pressing the Shop button on the upper right corner of the Apps menu. After logging into the Play Store using a Google or Gmail account, an enormous library of applications and games becomes available, with many of them offering free versions. This is also how applications are updated, and notifications will periodically appear requesting permission to update them.

If you already have an Android phone, you can quickly install your favorite applications by selecting the “My Apps” section from the upper left of the Play Store, then pressing the “All” tab which keeps track of any applications that you’ve used on any other Android device.

Chromium and Firefox

There are only a few browsers available for the Android platform, with

Firefox and Chromium being the most popular. Chromium has several features that aren’t available in Firefox, such as touchpad swiping and keyboard shortcuts. Either browser can be synchronized with other devices to automatically download any bookmarks that were made on the same Google account. Youtube videos should be played using the native Youtube app in order to leverage the video decoding hardware.

Gmail and Google Drive

More than one Google account may be added in the Settings app, and both Gmail and Google Drive offer an account dropdown as a way to easily switch between multiple accounts. Google Drive includes word processing, spreadsheet, presentation, forms, and drawing applications which are intended as a lightweight alternative to Microsoft Office. The advantage of using these cloud-based applications is that if there is an issue with the storage device, and the operating system needs to be restored, no documents will be lost, since they are stored on the cloud rather than a local hard drive, as previously mentioned.

XBMC and MX Player

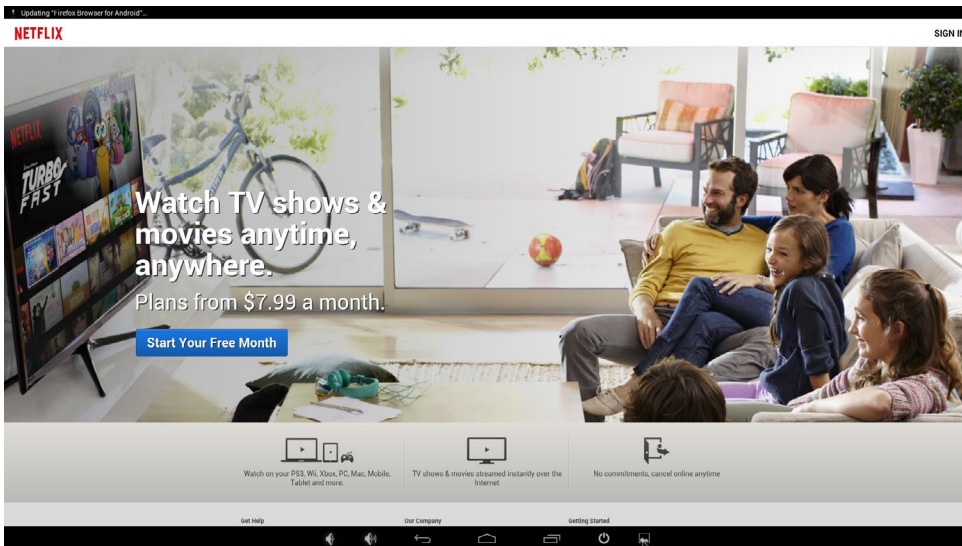
What set-top box would be complete without a killer media player? Both

XBMC and MX Player offer top-notch 1080p viewing experiences, with advanced features such as multi-language subtitles, audio visualizations, network streaming and Samba file sharing. XBMC also supports add-ons that offer TV channels and other media services.

MX Player is arguably the best audio and video player available for Android, and is able to play nearly any kind of media file. It also has a full screen option that hides the Android buttons at the bottom of the screen automatically, which remain visible while watching videos in XBMC. Make sure to turn on all of the HW+ decoding options in the Settings menu.

tTorrent

Torrents are automatically associated by Chromium with the tTorrent app, which is a peer-to-peer network downloader suitable for large media files. Completed files are, by default, saved to the /mnt/sdcard/Downloads directory, with the option to choose an alternate location. Keep in mind that tTorrent is extremely resource-intensive, and high disk activity may slow down the Android interface. It’s usually best to move any torrents or download activities to a second ODROID running Linux services, and access the completed files using the Samba sharing built into ES File Explorer.



Netflix is an essential part of any set-top box!

Netflix

Netflix is what every entertainment system needs, and the ODROID can run it in full 1080p on your giant living room screen. This subscription-based service not only offers a massive catalog of movies, there are many original shows that can only be seen on Netflix. The Netflix Android app is easy to use, with a similar interface to the desktop version, and movies can be searched using an actual keyboard instead of picking letters with a tiny remote from an alphabetic list like a 1980s arcade game.

Youtube and TED Talks

The Youtube app runs extraordinarily well, and can play 1080p streaming

videos at full screen resolution without playback glitches. The TED Talks app provides hours of educational and inspirational videos on a huge variety of topics, presented by a non-profit organization. For more information on TED Talks, visit <http://www.ted.com>.

Hangouts

The ODROID, when paired with an external camera such as the Logitech C920, supports 1080p video in Hangouts, just like a desktop PC. Either the internal camera microphone or an external microphone headset may be used.

Angry Birds

One of the most popular mobile games of all time is even more fun on

the ODROID! The developers adapted the game for use with a keyboard and mouse. Holding down the left mouse button is the identical to dragging on a touchscreen. Free versions of Star Wars, Star Wars II, and Space are already installed, and other versions, including more recent releases, can be downloaded from the Google Play Store.

Mupen64 Plus and Retroarch

Pocket Rocket can turn an ODROID into a family gaming super-console with these emulator apps. Both Mupen64 Plus and Retroarch support up to 4 simultaneous controllers, and support tens of thousands of console games. The Google Play Store includes many free emulators, and I tested a few of them for both game and controller compatibility before deciding to include Mupen64 Plus and Retroarch.

Retroarch is a multi-system emulator, and comes with many console cores that range from Atari 2600 to SNES to Sega Genesis to PlayStation Portable and PlayStation 1. It's an open-source app that combines many generations of retro and modern console video gaming consoles into a single system, and allows continuous hopping from console to console without having to reconfigure the controllers. Please refer to the July issue of ODROID Magazine for instructions on connecting Xbox 360 wireless controllers to Retroarch, which I have

Who doesn't like Angry Birds? Pocket Rocket and Couch Potato come with 3 versions of this highly addictive game.



found to be the most compatible controllers for use with Android.

Mupen64 Plus is an excellent Nintendo 64 emulator, and was the least buggy of all those that I tried. It has a great interface for setting up controllers quickly, and comes with several pre-built controller profiles, including Xbox 360 wireless. Both Retroarch and Mupen64 Plus allow use of the keyboard as a controller, as well as an alternative touch-screen controller interface, which is ideal for use with an ODROID-VU external touchscreen.

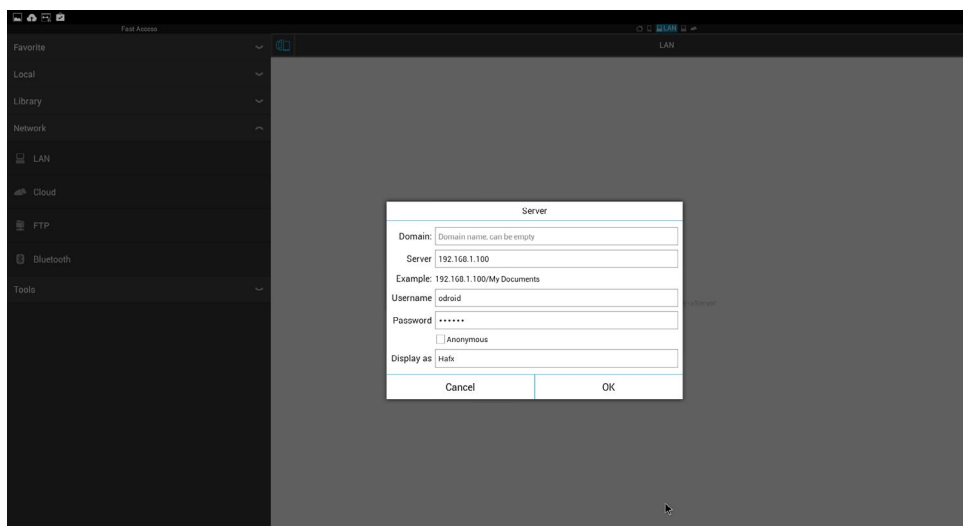
Pandora and PC Radio

Pandora is a subscription-based app that chooses music for you based on your past preferences and personal taste. If you want to have a continuous non-stop music mix for a party or dinner, put on a Pandora station and press the power button on the top of the U3 (or keyboard power button) to turn off the screen. PC Radio is an Internet Radio app that includes broadcasts from all over the world.

ES File Explorer and Ultra Explorer

These file explorers are more full-featured than some desktop explorers, offering Samba sharing, built-in zip extraction, application file associations, and multiple copy/paste blocks. Ultra Explorer is available as an open-source project where developers are encouraged to contribute their own modifications at <http://bit.ly/1AzAeVG>.

I used ES File Explorer's LAN tab to connect with my media server, so that when I click on media files, they automatically launch in MX Player, and MX Player streams them over the network. Additional local hard drives may be hot-plugged into the USB ports and accessed using ES File Explorer's Local tab, which lists each USB drive after it



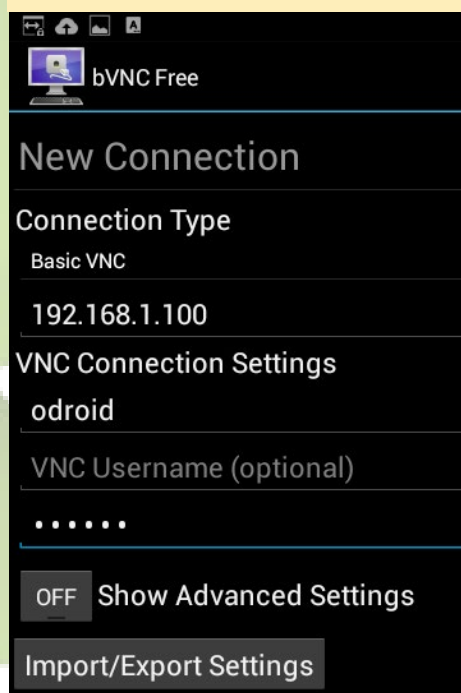
ES File Explorer can be used to connect Samba drives for network media streaming.

automatically mounts them.

bVNCFree and JuiceSSH

I use Android as my main operating system primarily because of bVNC-Free. With this excellent VNC application, I can login to any of my Linux computers and use the remote Ubuntu desktop as if it were a native Android app. By setting up a VNC server on the second ODROID using Vino, I need only a single monitor, keyboard and

bVNCFree can connect to any Windows, Linux, Android or OSX machine.



mouse, which can be shared between many computers. JuiceSSH provides the same functions as the Linux version of SSH, with a nice graphical interface to organize the recently used list.

Terminal Emulator and BusyBox Free

Terminal Emulator is an Android version of BASH terminal, and can be used in conjunction with BusyBox Free to perform all of the Linux administrative tasks that are not included with Android, such as mount, grep, ping and uname. For more information on the BusyBox commands, please refer to the official Wikipedia page at <http://bit.ly/1zwVNep>.

Jota+

Jota+ is a standard text editor, with nearly all of the formatting and editing features that would normally be included in a desktop text editor such as TextPad. Jota+ can be also used for web development by using the ES File Explorer to associate .html and .php files with Jota+. I created a Samba share on my ODROID web server to allow remote editing of the website files, and often use Jota+ to make quick changes to files via the Samba share without the overhead of starting up a full IDE.

Screen Orientation and full!screen

These two applications enable total control of the Android desktop. Screen Orientation will lock the screen in either Landscape or Portrait mode, depending on the orientation of the monitor. The full!screen application temporarily hides the bottom Android menu until one of the bottom corner hotspots is clicked.

Bluez IME Settings and Joysticks

Bluetooth controllers, such as the Wiimote, can be configured as input devices using Bluez IME Settings, so that they can be used to mimic keyboard presses. Xbox 360 controllers will immediately connect upon being plugged in to Pocket Rocket, and can be used to navigate the desktop menu, launch applications, and type using the on-screen keyboard.

ODROID Updater and ODROID Utility

The official Hardkernel version of KitKat Pocket Rocket is still in Beta as of August 2014, but the ODROID Updater can be used to import the latest improvements and bug fixes from Hardkernel as KitKat moves toward its final release. Follow the sequence of on-screen buttons to automatically install the system updates, which typically require a reboot. The ODROID Utility program is designed to adjust boot variables such as 720p and 1080p screen resolution, initial screen orientation, and CPU governor settings. Additional boot options, such as whether to blink the alive led, can be tweaked by editing the file /mnt/sdcard/boot.ini.

Installing Custom Applications

Although Pocket Rocket is loaded

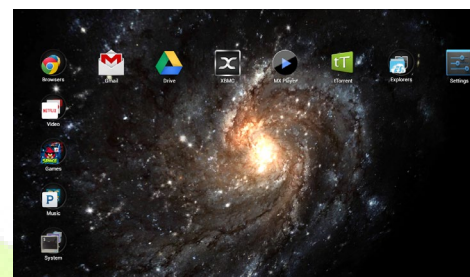
with third-party features, it can also serve as a general-purpose Android test environment for your own applications, by using the Android Debug Bridge (ADB) software. Install the ADB client on any host Linux machine by selecting it from the Synaptic Package Manager, plug the ODROID's micro-USB port to the host machine with a USB cable, and open an ADB connection by typing the following in Terminal on the host machine:

```
$ su
# adb remount
# adb shell
```

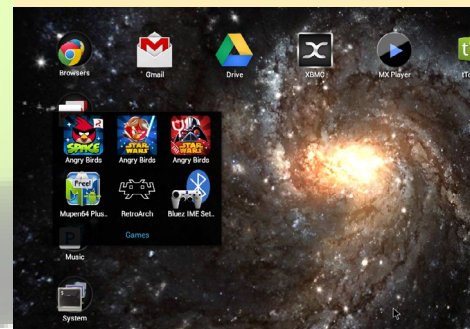
This will open an SSH-like session on the Android machine, permitting access to the command line on the remote machine. Files can be transferred via ADB to Pocket Rocket using the command:

```
# adb push <local directory> \
<remote directory>
```

For more information on using ADB and other ways of connecting to the ODROID from a host machine, please refer to <http://bit.ly/1xC42wk>.



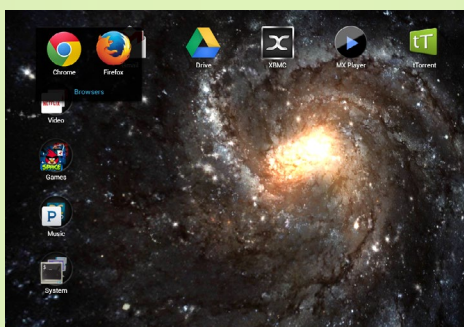
The Pocket Rocket desktop with a swirling Milky Way galaxy animation



The Games menu is where I spend most of my time



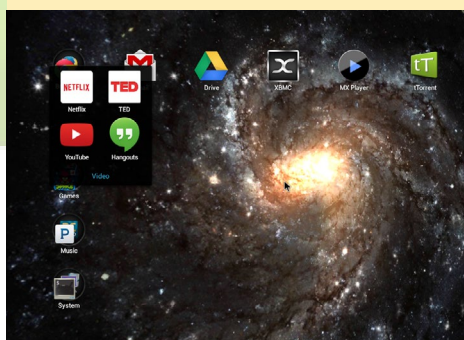
Chill out with Pandora or the worldwide Internet Radio using PCRadio



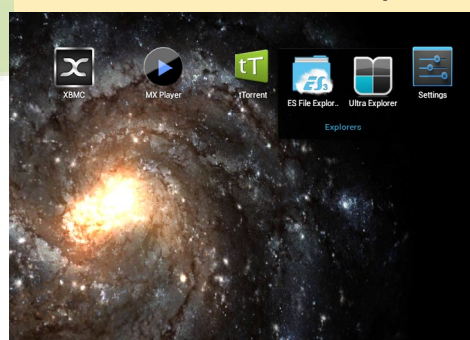
Chrome and Firefox are the most popular browsers available



The System menu lets you access protected files and other network computers



The Video menu also contains Google Hangouts and TED Talks



ES File Explorer and Ultra Explorer are two full-featured Android file explorers

MEET AN ODROIDIAN

BO LECHNOWSKY: EXPERT MAKER AND INSPIRATIONAL INVENTOR

by Bohdan Lechnowsky
edited by Rob Roy

Please tell us a little about yourself.

I was born in Omaha, Nebraska USA to Ukrainian and German immigrant parents. I moved to Ukiah, California USA after I had been married for 7 years and had one son and a daughter on the way. I now have two additional daughters for a total of 4 children, and have been married for 23 years. In 1994, I started an Amiga Computer store. In 2001, I started a technology consultancy and we now employ about a dozen people, mostly technicians.

How did you get started with computers?

Prior to computers, I had an interest in electronics. I received my first soldering iron at age 7. My parents lived very frugally, so I couldn't afford to buy new electronic components. Therefore, I had to take apart discarded electronics to sal-

vage what I could. I spent some time in the public library reading books on electronic circuits to learn more about it. When I was 12, my school purchased a few Apple II computers. I had to stay after school to use them which meant that I missed the bus and had a 2.5 mile walk home. Again, I had to teach myself programming from library books, and a fair amount that I taught myself. I got my own Commodore 64 at age 15. In school and in the university, I worked on all types of computers, mostly programming.

What would you consider to be your favorite ODROID?

Bo Lechnowsky, one of the Editors of the ODROID Magazine, playing his woodwind instrument outdoors



My favorite ODROID is the U3 as it is a very powerful and capable computer with a low price point, and there are a lot of U3-sized accessories like the Show, I/O Board and UPS Board, all of which I have been working with a lot lately.

You're an expert in the Rebol language, and often write articles that showcase the simplicity of the language. What software have you produced with Rebol?

My entire company runs on Rebol soft-

This is where all of the mad computer science experiments happen!



ware I developed starting in 2003. Everything from inventory and time logging to invoicing, accounts payable and receivable, and graphical real-time tracking of our progress toward our goals. In addition, I've developed super-efficient surveillance cameras that capture and process video on-camera at 1920x1080 resolution @ 30fps, and I've written the monitoring software that can be used to keep an eye on what's going on, and review what's already happened. For another project, I converted a client's medical recruiting database from a PICK mainframe to a Rebol/SQL solution that has now been in constant use since 2001. I could go on, but I'd seriously have to write a book to list all my projects written in Rebol.

What hobbies and interests do you have apart from computers?

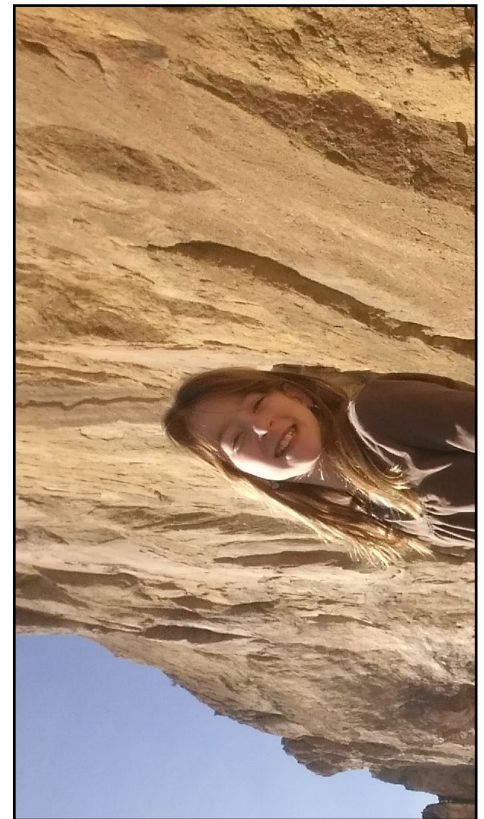
I like to say that if you need a hobby, I'd be happy to give you one of mine. It would be easier to list the hobbies I don't have. Some of the hobbies that I've been spending more time on lately have been raising a family, playing keyboards with my church's band, developing new technology products and business, working on the mechanics and body of my truck, traditional blacksmithing, participating in outdoor activities (mountain biking, camping, barbeque cooking, World Human-Powered Speed Championships), traveling (including three mission trips to Ukraine for a total of almost half a year), linguistics, and editing.

Are you involved with any projects?

Currently, I have been contracted to provide a solution to help improve food production by real-time processing and automation. In this project, we are using a U3 with Show and IO Shield boards coupled with sensors including GPS and lasers, and Rebol handles all the logic. Another project I just completed was setting up a video surveillance server using a U3 and an external storage device. I'm currently trying to find the time to upgrade my 1978 Ford F150 4x4 pickup truck to have an ODROID-powered brain allowing me to do things such as replacing the entire dashboard with digital touchscreens and Show boards, starting and stopping the engine with my smartphone, recording and checking fuel levels and performance data, integrating GPS navigation, and many automation features. Unfortunately, my contract projects have to take priority over my personal ones, like upgrading my truck. If I have a chance to get around to it, I'll be writing an article on my truck project for a future issue.

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

I haven't looked into the technical requirements of any of these, but as long as I'm allowed to make a wishlist I would love to see a future Hardkernel board with the newest Samsung SoC (looks like I'll get my wish with the XU3). It would also be amazing to have even a single laptop-style memory socket for expansion purposes, and running the



Hiking Smith Rock State Park in Oregon with the family

ODROID with PoE (Power over Ethernet). Of course, lots of USB ports are always a big plus (I know some boards already have quite a few). Maybe built-in bluetooth? With this said, my opinion is that Hardkernel does a great job of balancing power and features with price.

What advice do you have for someone wanting to get started with ODROID programming?

One of the big hurdles I think many

Bo's computer science laboratory is so big, it needed a second page to show everything!





Part of my business, Respectech, with the floating expanded metal floor and LED lighting underneath.

people face is setting up a development environment. With this in mind, you can't beat C programming as gcc (a C compiler) is already in almost every distro, and nearly every platform uses C (including embedded devices like Arduino). One would think that bash would also be a good option as it is also included in nearly every distro. However, my opinion is that bash goes overboard syntactically and is usually cumbersome with which to develop solutions. My

favorite, of course, is Rebol. It runs on nearly every platform without any code changes, it is a single file download of less than 1MB, and simple things are simple to do, while complex things are still possible. Not only that, but efficient dialects are easy to develop to allow for building complex systems easily, which reduces future development and debug-

your project. For me, Rebol has the flexibility and capabilities to handle almost every project I throw at it, and allows me to develop solutions in a fraction of the time it would take with other languages. And now that Red is maturing, it is going to be a great alternative option that will have nearly no learning curve for developers familiar with Rebol.

gging efforts. The language you use should be based on the end goals of

You can visit Bo's website at <http://www.respectech.com>.

Some of the "heavy lifting" we're doing to get my 1978 Ford F150 4x4 ready to have ODROIDs installed and controlling the electronics. My father-in-law helped with this step.



I can't tell you exactly what this project does (it's still top secret), but I can tell you that it involves a U3, a SHOW and an I/O Shield, among other things.

