I/O Shield access • Install and learn the basics of Rebol now!

ODROID Magazine

Year One
Issue #2
Feb 2014

Turn your ODROID into a:

GIANT TABLET

And create the ultimate Android experience!

High Performance computing at home

Creation and fine granular control

- Estimating Radio net interference:
  Using Java mutithreading
- Start programming right away!
- USB Gadget drivers
  Dominate your device ports to do everything
- Meet an Odroidian
  Mauro Ribeiro, Senior Software engineer
- Linux Gaming - emulators
- The art of multi boxing
- 811.02ac router with odroid XU
  Get up to 433mbit/sec

A service to the world-wide ODROID and Open Source communities, Hard Kernel is proud to present its newest contribution to ARM technology: ODROID Magazine, a free monthly PDF e-zine!

This cutting-edge online publication brings you the latest ODROID news, as well as featured articles from the expert community that has grown around the amazing ODROID family of micro-powerhouse computers.

Intended for all levels of expertise from beginner to guru, ODROID Magazine features definitive guides for new owners, with easy-to-follow steps in setting up your ODROID, installing operating systems and software, and troubleshooting common issues. For more technical users, each month will feature expert tips, hacker discussions, cutting-edge projects, and technical articles to explore new ways of making your ODROID even more versatile.

Hard Kernel's ODROID Magazine is an ideal opportunity for our community to come together to share and contribute articles, so that everyone can be successful with their ODROID.

Each month, a series of article topics will be posted for consideration, and all community members are encouraged to send submissions in exchange for monthly rewards for those selected for publication.

The best articles are those that walk the reader through complex concepts and procedures in a simple-to-read format. At least one picture or graphic per article is required, and should be between 500-2000 words.

In this issue, we show you how to turn your ODROID into an enormous 42" touchscreen Android tablet, suitable for kiosks, digital signage, gaming, accessibility, and just plain fun. You've never played Fruit Ninja like this!

ODROID Magazine

Rob Roy, Chief Editor

I am a computer programmer living and working in Silicon Valley, CA, USA, designing and building websites such as Vevo, Hi5, Dolby Laboratories and Hyundai. 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 own a lot of ODROIDs, which I use for a variety of purposes, including media center, web server, application development workstation, and gaming console.

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-U2. I have deep experience with many unique operating systems.

Bruno Doiche, Art Editor

Spent his last vacation doing nothing. And didn't manage to do the half of it.

ODROID Magazine, published monthly at http://magazine.odroid.com, is your source for all things ODROIDian. • Hard Kernel, Ltd. • 704 Anyang K-Center, Gwanyang, Dongan, Anyang, Gyeonggi, South Korea, 431-815 • Makers of the ODROID family of quad-core development boards and the world's first ARM big.LITTLE architecture based single board computer.
Join the ODROID community with members from over 135 countries, at http://forum.odroid.com, and explore the new technologies offered by Hardkernel at http://www.hardkernel.com.

CONVERTING A MONITOR TO A GIANT ANDROID TABLET

Justin Lee and Charles Park

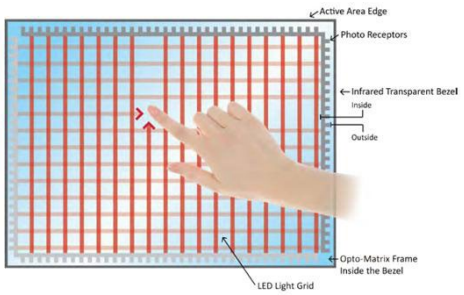
Touch screens are common in devices such as smartphones, game consoles, all-in-one computers and tablets. They also play a prominent role in the design of digital appliances such as digital signage, Point of Sale (POS) systems, satellite navigation devices, mobile phones, video games and some e-books.

The Android OS, one of the main operating systems for the ODROID, has an intuitive user interface designed for use with a touch screen. This article describes how to use an ODROID to change any monitor or TV into a giant Android tablet.

Infrared vs Capacitive touch screen

Touch screens primarily use either infrared or capacitive technology. Capacitive touch screens are more popular for smartphones and tablets, but are also more expensive, especially when the screen size is larger than 20 inches. A capacitive screen can only be activated with an exposed finger (no gloves or pointers), and can experience operational difficulties if the monitor is not correctly mounted into a metal housing due to the electrical field. Considering its ease of use and lower cost, the infrared-type touch screen is better suited for this project.

Infrared(IR) Grid touch screens

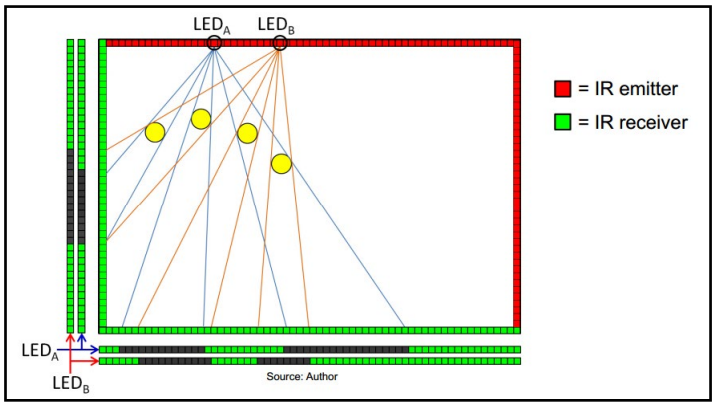


Principle of IR (Infrared) touch screen

An infrared touch screen uses an array of X-Y infrared LED and photodetector pairs around the edges of the screen to detect a disruption in the pattern of LED beams. These LED beams cross each other in vertical and horizontal patterns, which helps the sensors pick up the exact location of the touch. A major benefit of the infrared system is that it can detect essentially any input, including a finger, gloved finger, stylus or pen. It is generally used in outdoor applications and point of sale systems which cannot rely on a conductor (such as a bare finger) to activate the touch screen.

Unlike capacitive touch screens, infrared touch screens do not require any patterning on the glass, which increases durability and optical clarity of the overall system. However, infrared touch screens are sensitive to dirt and dust that can interfere with the IR beams, and suffer from parallax in curved surfaces and accidental touch notifications

if the user hovers his/her finger over the screen while searching for the item to be selected.



Principle of IR (Infrared) Multi-touch screen

How to Choose an Infrared touch screen

You can make your own touch screen by following guides found on the Internet, but it is not easy to implement the complex multi-touch algorithm and well aligned IR emitter/receiver pairs.

Before purchasing an infrared touch screen, it's important to evaluate its compatibility with Android. It must meet at least one of the following requirements:

1) Is your touch screen listed in the Linux multi-touch compatibility table?
`http://lii-enac.fr/en/architecture/linux-input/multitouch-devices.html`
If yes, it will be very easy to activate your touch screen.

2) Is your touch screen Windows 8 compatible?
If yes, you need to add a few lines in the Kernel driver and an input configuration file.

3) Does your touch screen manufacturer supply specific Android driver source code?
If yes, you need to follow their porting instruction.

A touch screen which meets the first requirement was not available in our local Korean or Chinese markets. Some touch screen manufacturers in China offered to supply the driver source code for their products, but the sample code was not useful in the real world. The best alternative was a touch screen that supported Windows 8 HID-compliant Plug & Play.

It's important to check whether the touch screen is really Windows 8 compatible or not. True Plug & Play devices do not require a separate driver to be installed on a Windows PC. If the touch screen needs a specific device driver, it is not natively compatible with Windows 8 and will be less likely to work with Android.

There are 4 steps to using the touch screen with the Android OS in a nutshell:

1) Get the Vendor ID and Product ID from the touch screen USB interface.
2)Modify the files hid-ids.h and hid-multitouch.c, both located in kernel/drivers/hid/
3) Build the kernel with the HID-MULTITOUCH option enabled, and transfer the kernel image to the ODROID.
4) Create an IDC (Input Device Configuration) file.

Step 1: Check the VID and PID

Plug the touch screen into any Linux PC, then find the VID and PID by typing `lsusb` in the terminal as below right.

To determine which device entry is associated with the touch screen, list the devices before connecting the touch screen, then list them again after connecting it. The new entry will correspond to the touch screen device.

During our project, when the 23-inch touch screen was connected, an entry of VID:03FC, PID:05D8 appeared in the device list, which represents a touch screen made by Elitegroup Computer Systems. With the 42-inch touch screen connected, an entry of VID:1870, PID:0119 appeared to represent a touch screen from Nexio Co., Ltd. Other touch screens will report different VID, PID and vendor information.

```
odroid@odroid:~$ lsusb
Bus 002 Device 068: ID 05e3:0610 Genesys Logic, Inc. 4-port hub
Bus 002 Device 002: ID 8087:0024 Intel Corp. Integrated Rate Matching Hub
Bus 002 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 006 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 005 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 004 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 003 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 001 Device 002: ID 8087:0024 Intel Corp. Integrated Rate Matching Hub
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
odroid@odroid:~$ lsusb
Bus 002 Device 073: ID 03fc:05d8 Elitegroup Computer Systems
Bus 002 Device 068: ID 05e3:0610 Genesys Logic, Inc. 4-port hub
Bus 002 Device 002: ID 8087:0024 Intel Corp. Integrated Rate Matching Hub
Bus 002 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 006 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 005 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 004 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 003 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 001 Device 002: ID 8087:0024 Intel Corp. Integrated Rate Matching Hub
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
odroid@odroid:~$ lsusb
Bus 002 Device 074: ID 1870:0119 Nexio Co., Ltd
Bus 002 Device 068: ID 05e3:0610 Genesys Logic, Inc. 4-port hub
Bus 002 Device 002: ID 8087:0024 Intel Corp. Integrated Rate Matching Hub
Bus 002 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 006 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 005 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 004 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 003 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 001 Device 002: ID 8087:0024 Intel Corp. Integrated Rate Matching Hub
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
odroid@odroid:~$
```

Reading the VID and PID of the USB touchscreen using a Linux PC

Step 2: Modify hid-ids.h and hid-multitouch.c

After downloading the appropriate Android kernel source from dn.odroid.com, navigate to the kernels/drivers/hid/ directory, then add the VID and PID to the end of the hid-ids.h header file.

```
#define USB_VENDOR_ID_PRIMAX0x0461
#define USB_DEVICE_ID_PRIMAX_KEYBOARD0x4e05

/* Elitegroup Computer Systems */
#define USB_VENDOR_ID_ELITEGROUP     0x03fc
#define USB_DEVICE_ID_ELITEGROUP_TOUCH  0x05d8

/* Nexio Co., Ltd */
#define USB_VENDOR_ID_NEXIO      0x1870
#define USB_DEVICE_ID_NEXIO_TOUCH    0x0119

#endif
```

Example PID and VID values added to `kernels/drivers/hid/hid-ids.h`

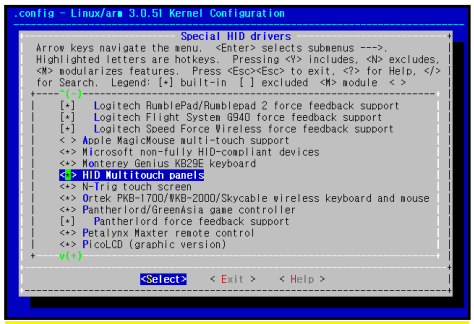
Also, add the new ID in the hid-multitouch.c source file. It must be placed in the hid_device_id mt_devices structure define.

```
static const struct hid_device_id mt_devices[] = {

/* Elitegroup Computer Systems */
{ .driver_data = MT_CLS_DEFAULT,
        HID_USB_DEVICE(USB_VENDOR_ID_ELITEGROUP,
                       USB_DEVICE_ID_ELITEGROUP_TOUCH) },

/* Nexio Co., Ltd */
{ .driver_data = MT_CLS_DEFAULT,
    HID_USB_DEVICE(USB_VENDOR_ID_NEXIO,
                       USB_DEVICE_ID_NEXIO_TOUCH) },

/* 3M panels */
{ .driver_data = MT_CLS_3M,
HID_USB_DEVICE(USB_VENDOR_ID_3M,
USB_DEVICE_ID_3M1968) },
{ .driver_data = MT_CLS_3M,
HID_USB_DEVICE(USB_VENDOR_ID_3M,
USB_DEVICE_ID_3M2256) },
{ .driver_data = MT_CLS_3M,
HID_USB_DEVICE(USB_VENDOR_ID_3M,
```

Example of adding the touch screen ID to `kernels/drivers/hid/hid-multitouch.c`

Step 3: Build the kernel with the HID-MULTITOUCH option enabled

Type "`make menuconfig`" to configure the kernel, then go to `Device Drivers > HID Devices > Special HID drivers > HID Multitouch panels` and select it as an embedded driver (*), as seen below.



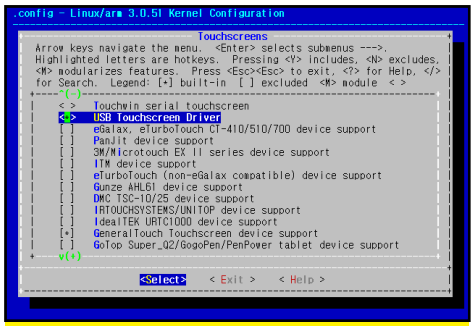
Configuring the touch screen with an embedded driver in the kernel configuration.

Then, set another two options as an embedded driver, indicated with an asterisk (*)

`Device Drivers > Input device support > touch screens > USB touch screen Driver`

and:

`Device Drivers > Input device support > touch screens > GeneralTouch touch screen device support`



Configuring the touch screen with an embedded driver in the kernel configuration

Save the Kernel configuration and compile it to make a zImage.

Transfer the zImage to your ODROID via fastboot protocol in the u-boot.

Step 4: Create an IDC (Input Device Configuration) file

If you don't make a proper IDC file, the resolution of the touch screen will not match the HDMI resolution. Create a plain text file as below:

```
touch.deviceType = touch-
Screen
touch.orientationAware = 1
device.internal = 1
keyboard.layout = qwerty
keyboard.characterMap = qwer-
ty2
keyboard.orientationAware =
1
keyboard.builtIn = 1cursor.
mode = navigation
cursor.orientationAware = 1
```

The file name must be `Vendor_xxxx_Product_yyyy.idc (xxxx: Vendor ID, yyyy: Device ID).` I made two files for Elitegroup and Nexio. The filename is case sensitive.

`Vendor_03fc_Product_05d8.idc`
and
`Vendor_1870_Product_0119.idc`

Copy the IDC files to your ODROID with the below commands.

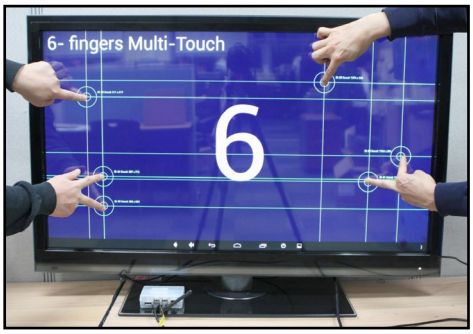
```
adb remount
adb push Vendor_03fc_
Product_05d8.idc /system/
usr/idc/.
adb push Vendor_1870_
Product_0119.idc /system/
usr/idc/.
adb reboot
```

How to Attach the 42-inch Touch Panel to the TV Screen.

Carefully attach the touch screen to align the viewing window.

Preparing the touch screen: Attaching the double-sided tape to the touch screen frame

Test the touch screen. Our touch screen could detect up to 6 points.



Testing the touch screen - 6 points guarantee the sensor integration with the software.

This smaller 23-inch touch screen panel came with 4 velcro belts and it was relatively easy to assemble.

23-inch touch screen panel with velcro belts playing Fruit Ninja on android.

Conclusion

Besides gaming and personal use, the ODROID is ideal as the core computing device for kiosks, digital signage, human interface research, and more, because of its high performance computing power, relatively low cost, and open platform which allows modifications such as this touch screen.

A video of the results of this project, is on `http://www.youtube.com/watch?v=HDsnuxchxtU`, and for more giant action, `http://www.youtube.com/watch?v=n8_cV_NeWQ8`.

INSTALLING ANDROID ON AN ODROID

THE MAD SCIENTIST CHRONICLES CONTINUE

Bohdan Lechnowsky

This moment finds you in your familiar laboratory, hunched over your lab counter, illuminated by the soft glow of your plasma display, network indicator lights, catalyzing chemicals in various test tubes, and such. Before you lays a brand new ODROID-U3 which you are in the process of modifying into a combination superphone/world-domination device.

On another ODROID inches to your right plays an inspiring YouTube video of The Ben Heck Show while you hack old IDE ribbon cables and connect your U3 to various items on your worktop like batteries and miniature displays.

Upon completing the last solder connection, you look up with a menacing smile of accomplishment and say, "Flip the switch!" After a brief moment of inactivity, you remember that you still don't have a lab minion. Not only that, but there is no switch to flip. However, there is an on/off button on the U3!

You power on your U3, but then realize after another brief moment of inactivity that you don't have an OS on your SD or eMMC card. You decide that you should probably install Android as it is the best choice for the superphone portion of your invention, and you can run Linux in parallel with Android for the world-domination half of your invention, as explained in the January 2014 issue of ODROID Magazine. Handily, you printed out and bound the entire last issue of ODROID Magazine for easy reference. Sometimes, it's just nice to feel paper between your fingers.

After a few cobbled attempts at downloading the latest Android release for the ODROID, you actually slow down and study the proper way to do it.

As the U3 is – for most intents and purposes – identical to the U2, you follow the procedures for installation on the U2. You find the latest full version of Android for the U2 is the Android Beta 1.6, and the newer versions are simply updates to that version.

You download Android_Beta_1.6 for the U2 from `http://dn.odroid.com/Android_Beta_1.6/U2/` and write it to your SD/eMMC card as explained in the January 2014 issue of ODROID Magazine. Once it has completed, you plug the card into your ODROID and yell, "PUSH THE BUTTON!", this time remembering that you have neither a minion nor a flippy-type switch. You catch yourself cackling in glee as the ANDROID logo appears across your mini display connected to your U3.

You find instructions for updating to a newer version of Android on the odroid.com blog, and you make some changes to bring it up to the current version, like so:

Check that your OS version is higher than Beta 1.6

`(Settings > System > About Tablet)`

The Build Number has a date code, and should be April-2013 or later. This image is 24-April-2013.

Download the Android update file.

Visit `http://dn.odroid.com/Android_Beta_1.8.0/U2/` (for other models of ODROID, leave off the "U2" at the end). Select the SD or eMMC image, depending on the card that you have installed, and start the download.

Create the updater directory.

Create the directory `/sdcard/updater` using File Explorer that should be included in your Apps list.

Move the update file to the updater folder.

Copy the downloaded .zip file containing the Android Beta into the `/sdcard/updater folder.`

Start the update process.

`(Settings > About Tablet > ODROID )`

HIGH PERFORMANCE COMPUTING AT HOME

COMPUTE LIKE YOU NEVER DID BEFORE

Cooper Filby and Anthony Skjellum - Runtime Computing Solutions LLC

If this doesn't make you want to make a Voltron based cluster using five ODROIDs, we cannot be held responsible for the giant robots that may someday take over your hometown.

In the previous issue of ODROID Magazine, we discussed the benefits of using ODROIDs for High Performance Computing (HPC), in addition to various discoveries while working with XU+E's related to setting up our own cluster in a commercial setting. But what about the home user that wants to experiment with HPC? In part one of this multi-part series, we outline the setup and configuration of a basic "headless" cluster with the end goal of running parallel programs based on message passing, using the Message Passing Interface (MPI) parallel programming model in particular.

In subsequent articles, we plan to expand on this setup to build a cluster with a centralized head system or node, complete with services such as Puppet, NFS, and LDAP, a configuration that is capable of supporting a much larger number of ODROID systems (nodes) configured as an HPC cluster. For example, in part two, we will cover basic networking configuration of the head node, including dnsmasq, NAT, and adding more nodes to the cluster, while in part three (and further installments), we will cover additional services such as LDAP, NFS, Autofs and Puppet.

Installing the Operating System

There are a number of prebuilt Linux operating systems available for ODROID boards from http://dn.odroid.in. To get started, download the Ubuntu Server image for your ODROID model and extract the .IMG.XZ archived image using an archiving tool such as 7zip on windows, or by typing "xz" from the Linux command line. Finally, you can copy to the medium of your choice, such as an SD card or an eMMC module, using the "dd" command on Linux/OS X systems or the Win32DiskImager.exe for ODROID on Windows. For more detailed instructions on copying over the OS, please refer to Bohdan Lechnowsky's article titled "Installing an OS on an ODROID" from the January 2014 issue of ODROID Magazine. We recommend using the eMMC modules available from Hardkernel for better performance, but SD cards work well too.

Connecting to your Odroid and User Configuration

Since we opted to use the Ubuntu Server image for our ODROIDs, we can connect to our XU-E systems (we'll call them nodes for simplicity from now on) via the ssh protocol using Terminal (or Putty if running Windows) in order to continue setting up our cluster. Because of potential initial hostname and MAC address conflicts that we will resolve in the next section, we will need to boot the first ODROID and set a few settings before starting the second.

[Editor's Note: If one is available, a development machine running Linux or Windows is recommended to more easily setup and reboot the cluster, troubleshoot hardware problems, and other necessary debugging. An alternative to using a separate computer is to plug a USB keyboard and HDMI cable into the first ODROID and use it directly to bootstrap the cluster instead of via SSH as described in the next few paragraphs. Press Ctrl-Alt-F1 to use the framebuffer console if X11 is not running.]

In order to connect to your ODROID, you'll need to discover the hostname or IP address of the board.

For the Ubuntu server image we used on our XU+E cluster, the default hostname is "odroid-server", while for other images we've used, it's been "odroid". Most home networks should support DNS by default, which will allow you to connect simply by the hostname. If this fails, you can alternatively connect using the IP address assigned to the ODROID by your router instead. If neither of the hostnames resolves for you, check your router's lease table to search for the IP address, often labeled as the DHCP client table in the router's admin panel.

Since we used identical copies of the same image on both nodes, by default they had a hostname conflict, which we resolved by bringing them online one at a time, then changing the individual network settings. If you don't have access to the router's admin panel, you can also make use of the nmap command to scan your network for hosts to find the ODROIDs, if you know your network information. For example: "nmap 192.168.1.0/24". Look for a host that has port 22 open.

Power on one of the ODROIDs, then enter "ssh odroid@ubuntu-server" (or "ssh odroid@xxx.xx.xx.xxx", if using the IP address) in the Terminal or Putty window of the host computer, which will establish a secure connection to the ODROID. To login, type "odroid" as the password.

Once the command prompt appears, you may want to run "sudo apt-get update && sudo apt-get upgrade" to ensure that your OS is up to date. Furthermore, we recommend you run the "passwd" command and change the password for the odroid user to something a little more secure, or creating new user accounts with the "adduser" command, such as by running "sudo adduser kilroy". (Generally speaking, do three things key with your node passwords: make them long, make them hard to guess, and store it in a secure location.)

Configuring Networking

Before getting both ODROIDs online, we need to change a few settings as to eliminate hostname and MAC address conflicts that may occur on your home network with an ODROID cluster. To change the hostname, we will need to edit two files, /etc/hostname and /etc/hosts, changing "odroid-server' to the hostname of your choice and rebooting the machine so the changes take effect. For the purposes of this article we will use odroid-server0 and odroid-server1 to refer to the first and second ODROID respectively. Alternatively, if your operating system supports it, you can also type "sudo odroid-config" to change the hostname. You can use other names of your choice; they have to be unique to each node.

The MAC address conflict was a subtle issue that we encountered when we first set up multiple ODROID XU+E's. We found that, by default, the onboard ethernet devices all shared the same MAC address, which made it impossible to work on a single ODROID if multiple were powered online and on the same network. If the two ODROIDs you're working have identical MAC addresses, there are two straightforward ways to resolve this: 1) configure one (or both) of the ODROIDs to use a different MACaddress, or 2) setup USB ethernet dongles, which should all have unique MAC addresses. The specific values you choose really don't matter, as long as you keep them unique on your Local Area Network (LAN).

To change the MAC address of the onboard device, edit `/etc/network/interfaces` with your text editor of choice, and add the line "hwaddress ether newmac", where newmac is an address in the format "b6:8d:67:7b:cb:e0" underneath the following labels:

```
auto eth0

iface eth0 inet dhcp
```

Then, reboot the ODROID so the changes take effect. Make sure to verify the new address using the ifconfig command. Alternatively, you can opt to plug your USB Ethernet adapters into the USB 3.0 slot, and then run "ifconfig -a | grep eth", which should yield a list similar to this:

```
eth0  Link  encap:Ethernet
HWaddr b6:8d:67:7b:cb:e0

eth2  Link  encap:Ethernet
HWaddr 00:13:3b:99:92:b1
```

By default, eth0 will be the onboard 10/100 ethernet connection, while the second ethernet device (in this case, eth2) will be the USB Ethernet Adapter. If only eth0 shows up, try reseating your

Yeah, we are serious when it comes to giant robots taking over your hometown, but Voltron always wins.

USB Ethernet adapter and/or verifying that it works on another machine. To set up the adapter for using DHCP on boot to get an IP address, we will need to modify `/etc/network/interfaces` and add the following two lines between the entries for auto lo and auto eth0:

```
auto eth2

iface eth2 inet dhcp
```

Use the appropriate ethernet device id previously found with ifconfig (in this case, eth2). Then, power down the ODROID, put the ethernet cable that was attached to the the onboard device into the USB ethernet adapter, and power the ODROID back on. If, for some reason, you aren't able to connect, try plugging the cable back into the onboard slot and verifying that the USB ethernet adapter is still showing up using the "ifconfig -a" command. It's also possible that the ethernet device ID itself has changed if the adapter is unseated, in which case you can update the /etc/network/interfaces file accordingly.

At this point, the ODROID should be configured and accessible on the network. Before heading on to the MPI section, configure the second ODROID using the same steps described above.

Message Passing Interface (MPI)

Now that we have two nodes configured appropriately, we can now start looking towards how we can execute HPC jobs on our two-node cluster. A parallel programming environment such as MPI helps you do this. MPI takes care of starting up the processes that make up the parallel programming model, and provides a standardized application programming interface (API) for those cooperating, communicating sequential processes to use to make the parallel program work. To accomplish this, we will make use of MPI, or Message Passing Interface, which provides an API that allows nodes to send and receive messages while processing jobs. A command called either mpirun or mpiexec will start all the processes needed across your ODROIDS under your control. There are two common open source MPI implementations available for download - MPICH and OpenMPI. For ODROID clustering purposes over Ethernet, both work equally well. Both of these MPI implementations are available through apt-get.

To install MPICH, run "sudo apt-get install mpich2", or run "sudo apt-get install openmpi-bin" to install OpenMPI as an alternative.

What you can do once you've loaded MPI:

1) Run test applications that use multiple cores on a single ODROID

2) Run example programs that use both ODROIDS and a total of 8 cores.

3) Learn how to build your own MPI programs.

In this article, we've focused on showing you how to do the first and the second approaches. You can read the example programs that come with OpenMPI and MPICH to learn more. There are also a number of excellent online tutorials and a few good books on programming MPI, such as "Using MPI" from MIT Press (one of us co-authored that book).

Building it Better

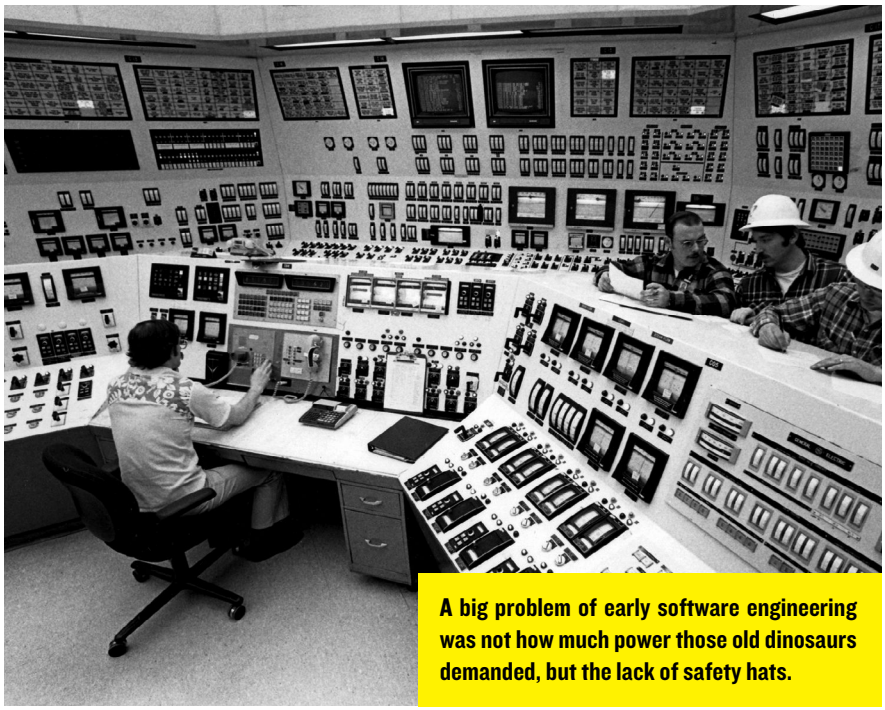
The content of this article represents just a fraction of what we will be able to do with our cluster down the line. While this setup is more than adequate for handling two nodes and only a few users, if we want to grow our cluster, we will want to make use of a dedicated head node to better handle a larger number of users and nodes. In addition to allowing us to hide cluster traffic from the rest of the network, this head node will also host services that will streamline cluster management, such as LDAP for user management, Puppet for content management, NFS for file sharing, and various networking services.

In part two of this series, we will begin to convert odroid-server0 into a proper head node.

FINE-GRAINED POWER CONTROL ON ODROID CLUSTERS

24 HIGH PERFORMANCE CORES FOR 35 WATTS

Kurt Keville



A big problem of early software engineering was not how much power those old dinosaurs demanded, but the lack of safety hats.

Our team has experimented recently with a number of new energy-saving techniques available to server system administrators and managers of High Performance Computing (HPC) clusters. Our specific experiences come in the form of testing a mini-cluster of ARM Cortex-A9 boards running the development Ubuntu version targeted at that platform.

In addition to the Hardkernel-recommended default install of the OS and concurrent headless server application suite (including MPI discussed on the last article), we installed cpufrequtils, WattsUp, and PowerNap and then updated the systems accordingly. These packages are mainly what we will discuss in this article, since they are where the bulk of our power / energy savings and reporting are attained.

The power savings are considerable in clusters that have significant reduced usage periods, and an aggressive power-saving schedule yielded no negative results in our findings. Even if you do not have big.LITTLE enabled apps (which we do not have on our U2 / U3 cluster), you can tune your performance requirements with rules written against packages like Slurm, which is currently one of the most popular queueing systems / schedulers in the HPC world.

The big takeaways from this project

Server hibernate is of decreasing utility since poweroff and reboot is just as quick in our usage. Governer invocation is now the standard.

The Powersave governor is of considerable value and should be used as a default in HPC clusters.

And manual processor affinity is no longer necessary if you are using modern kernels on your ODROIDs.

We have boot times on our eMMC fitted U2 and U3s of less than 10 seconds so there is no argument you can make for suspend / resume or hibernate / thaw that isn't better exemplified by an on / off solution or a scripted governor call in embedded HPC.

In an academic datacenter, you are quite often utilizing 100% of your most restrictive computational resources in your applications, or waiting for a job to be initiated. Because of conventional standard practices of "nice" level management and queueing system usage, most systems are fully in use if you organize your jobs correctly. We therefore investigated fully-on, fully-off and almost-off systems with an eye towards energy savings relative to the various options available to us that don't involve a full power-down.

Cold boots time for our demo cluster was less than 10 seconds and therefore validated this model. A more likely scenario for traditional datacenters, however, would include systems that are rarely powered down since they need to respond to requests quicker than a power-on would allow. For those scenarios we investigated suspend and resume and the novel PowerNap acpi modes.

Datacenter sysadmins have always had a way to do a "safe" managed shutdown of their servers in case of a power outage. The UPS vendor APC, in particular, has been a leader in this field when

they introduced their "SmartUPS" line. The original software application that shipped with these UPSes utilized a serial (RS-232) connection that would allow the system to initiate a shutdown script on a server when the UPS detected that it was no longer getting charged. Since the original releases there have been many improvements to this approach, affording substantial scripting flexibility to the sysadmin and introducing alternative hardware options, like USB and SNMP connections.

The open-source community has also added many new options to this functionality, progressively in the form of software support for the nut and upsd code bases (see http://www.networkupstools.org). We can now "hook" the invocation of this script into a user-defined usage, in our case tied to a drop in processor usage below 20%. NUT is compatible with Ganglia and Nagios in that it affords quite a few scripting options, so sysadmins familiar with adding ad-hoc functionality to their cluster management tools should have little problem integrating it with their custom setup.

Additionally, cpufrequtils and pmutils are very handy sets of tools that are compatible with the Linux kernel, at least as recently as 3.2. PowerNap lets you manage this via a script interface. You can lock your board into its highest speed setting by invoking "`cpufreq-set –g performance`" from the command line or by hooking it into PowerNap's config scripts if you prefer a degree of automation.

Note the ODROID U3 output above right, when your processors are set to "performance" mode (one report for each processor core normally).

When you set the U3 processor frequency to the stock maximum of 1.7 Ghz and run the workload at `http://openbenchmarking.org/result/1401173-UT-1309189UT21`, you can successfully peg the processors at close to maximum utilization. Our power peaked at 6.3 Watts and averaged 5.6 per node over the course of the benchmark suite run. It is interesting to note that while these particular benchmarks managed to keep the CPUs busy, it did not fully utilize available memory, which is also reflected in power usage.

```
root@odroid:~# cpufreq-info

cpufrequtils 008: cpufreq-info (C) Dominik Brodowski 2004-2009

Report errors and bugs to cpufreq@vger.kernel.org, please.

analyzing CPU 0: driver: exynos_cpufreq

 CPUs which run at the same hardware frequency: 0 1 2 3

 CPUs which need to have their frequency coordinated by software:
0 1 2 3

 maximum transition latency: 100.0 us.  hardware limits: 200 MHz
- 2.00 GHz

   available   cpufreq   governors:   conservative,   userspace,
powersave,performance

 current policy: frequency should be within 200 MHz and 1.70 GHz.
The governor "performance" may decide which speed to use within
this range.

 current CPU frequency is 1.70 GHz (asserted by call to hard-
ware).

 cpufreq stats: 2.00 GHz:0.00%, 1.92 GHz:0.00%, 1.80 GHz:0.00%,
1.70 GHz:93.58%, 1.60 GHz:0.01%, 1.50 GHz:0.00%, 1.40 GHz:0.00%,
1.30 GHz:0.01%, 1.20 GHz:0.00%, 1.10 GHz:0.01%, 1000 MHz:0.00%,
900 MHz:0.00%, 800 MHz:0.00%, 700 MHz:0.00%, 600 MHz:0.00%, 500
MHz:0.00%, 400 MHz:0.00%, 300 MHz:0.01%, 200 MHz:6.38%  (66)
```

cpufreq-info will show this 4 times, but if you go for powersave or conservative, get ready to see less active processors when idle.

top command sampling a lu.A benchmark

```
top - 09:00:58 up  2:52,  6 users,  load average: 2.41, 0.74, 0.29

Tasks: 154 total,   2 running, 152 sleeping,   0 stopped,   0 zombie

%Cpu(s): 99.1 us,  0.0 sy,  0.0 ni,  0.8 id,  0.0 wa,  0.0 hi,  0.1 si,
0.0

KiB Mem:   2071512 total,   884916 used,  1186596 free,   111884 buffers

KiB Swap:        0 total,        0 used,        0 free.   421392 cached
Mem

 PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COM-
MAND

6306 root      20   0   69660  43248    644 R 394.9  2.1   3:59.17 lu.A

6311 root      20   0    4708   1236    836 R   0.7  0.1   0:00.11 top
```

Some benchmarks represent usage as 400% of one process rather than 100% of 4 processes depending on whether they are initiated as MPI jobs. We tracked the power use using an FTDI-connected WattsUp meter. Clearly the XU-E is much better suited to this experiment not to mention the substantially better performance with the big.LITTLE architecture.

Our test setup

We wanted a demo micro-cluster that captured the power being used by six (6) ODROID-U2 computers. Those items were connected to our WattsUp meter, which allowed us to measure relatively small power fluctuations, since they aggregated. We captured this with the Linux utility described below. Additionally, we measured single node numbers on an ODROID U3. Baseline power usage on boot (note that boots put us into "ondemand" mode at initially 200 Mhz) was negligible. While a range from 200 MHz to 1.7 GHz were available speed options for our boards (without overclocking), as a practical matter we only used the lowest and highest speeds.

We measured the following scenarios; All boards at peak freq, all boards at minimum freq while retaining all cores, all boards with 2 CPUs on at min, and all boards with 1 CPU on at max.

PowerNap, in our conventional usage, controlled cpufreqd to dial down the CPU speed to its factory-enabled minimum when an idle period is detected. An idle period can be user defined, as well as other parameters associated with reporting, grace periods and other functionalities.

Our recommendations are to ignore hibernate in a server scenario; if you need immediate shutdown just power off underutilized cores at a board level. Rebooting takes ~10 seconds each in our configurations. While the U3 might not compete with the XU on FLOPs per Watt, it does on FLOPs per dollar. It is cheaper than an Intel Galileo and compares favorably in the openbenchmarking tests by that standard.

Moreover, it appears that power efficiency scales superlinearly with frequency and core count. So you if you have a power budget, and few time restrictions, you can script your cpufreqd to make that a priority, and aggressive manage your power use.

Governor setting versus watt utilization

Governor (freq)	Avg. Watts for 6 boards
Performance (1.7 Ghz)	33.6
Userspace (700 Mhz)	21.0
Powersave w / 2 core*	3.6
Performance w / 1 core	23.4

* via a PowerNap script that invokes "echo 0 > /sys/devices/system/cpu/cpu2/online" and cpu3. Actually, if you set any frequency below 700Mhz on the U3, it will turn off 2 cores.

Lower bars = less wattage usage

The SoC Drawer and SoX BoX clusters disassembled, those guys pack quite a punch

The power use logging setup on using the Odroid U3.

References

PowerNap Tutorial http://tinyurl.com/PowerNapTutorial

Slurm Integration http://tinyurl.com/SlurmIntegration

Power Use Logging http://tinyurl.com/WattsUpLinuxUtil

Gone are the days when notebooks came with Ethernet and COM ports. What do you do if you are stranded with nothing but a USB cable?

When developing applications for the ODROID, its often necessary to transfer files and commands from a main host PC to the development computer. This can be performed via FTP or SSH through the Ethernet port, but it can be also be done over USB, to simplify the setup. This article describes how to use the USB device port on your ODROID to communicate with your host PC. To get started, a micro-USB cable is required to make a physical connection between ODROID and your host PC.

By connecting the ODROID to a PC using its USB port, it can be "enumerated" to appear as another common USB device, depending on the type of communication desired:

Serial communication device

Network device

Mass storage device

You can also use the USB cable to emulate each of these data connections at the same time, as a multi-function link.

Installation

In order to use the Gadget drivers, first verify that the following modules are present on the ODROID system:

Mass storage emulation

```
/lib/modules/`uname -r`/kernel/
drivers/usb/gadget/
g_mass_storage.ko
```

Serial emulation

```
/lib/modules/`uname -r`/kernel/
drivers/usb/gadget/g_serial.ko
```

Ethernet emulation

/lib/modules/`uname -r`/kernel/drivers/usb/gadget/g_ether.ko

Mass storage + Serial + Ethernet

/lib/modules/`uname -r`/kernel/drivers/usb/gadget/g_multi.ko

If the drivers aren't initially present, the Kernel and driver modules need to be updated. Type the following commands into a Terminal window to upgrade:

```
$ mkdir update-kernel && cd update-kernel

$ wget http://builder.mdrjr.net/tools/kernel-update.sh

$ chmod +x kernel-update.sh

$ sudo ./kernel-update.sh
```

In the following code examples, the green characters show the Terminal window running on the ODROID, and the blue characters show the Terminal on the Host PC.

Emulating a mass storage device:

Storage over USB

Load the g_mass_storage module with a storage node which will be mounted on your host PC.

```
# mount

/dev/mmcblk0p1 on /media/boot type vfat (rw,nosuid,nodev,flush,umask=000)

# sudo umount /dev/mmcblk0p1

# sudo modprobe g_mass_storage file=/dev/mmcblk0p1
```

Launch dmesg and find the Mass_Storage Function gadget driver message.

```
… gadget: Mass Storage Function, version: 2009/09/11
… gadget: Number of LUNs=1
… lun0: LUN: file: /dev/mmcblk0p1
… gadget: Mass Storage Gadget, version: 2009/09/11
… gadget: userspace failed to provide iSerialNumber
…gadget:g_mass_storage ready
… g_mass_storage
gadget: high-speed config #1: Linux File-Backed Storage
```

Connect the ODROID's micro-USB cable to your PC and switch to using the Linux host.

On the Linux Host:

If a recent Linux distribution is installed, the device should be automatically mounted. Otherwise, follow these steps to manually add it to the host PC:

As root, launch dmesg and find the /dev/sdX device assigned to the ODROID:

```
.. scsi 56:0:0:0: Direct-Access Linux File-CD Gadget 0308 PQ: 0 ANSI: 2

.. sd 56:0:0:0: Attached scsi generic sg2 type 0

.. sd 56:0:0:0: [sdc] 262144 512-byte logical blocks: (134 MB/128 MiB)

.. sd 56:0:0:0: [sdc] Write Protect is off

.. sd 56:0:0:0: [sdc] Mode Sense: 00 00 00 00

.. sd 56:0:0:0: [sdc] Asking for cache data failed

.. sd 56:0:0:0: [sdc] Assuming drive cache: write through

.. sd 56:0:0:0: [sdc] Asking for cache data failed

.. sd 56:0:0:0: [sdc] Assuming drive cache: write through

.. sdc:

.. sd 56:0:0:0: [sdc] Asking for cache data failed

.. sd 56:0:0:0: [sdc] Assuming drive cache: write through

.. sd 56:0:0:0: [sdc] Attached SCSI disk
```

Then, mount the device and enjoy the file sharing:

```
$ sudo mount /dev/sdc /mnt/tmp

$ cd /mnt/tmp
```

To emulate a serial link :

Serial over USB

Load the g_serial module on the ODROID:

```
# sudo modprobe g_serial
```

Launch dmesg and find the Serial Function gadget driver ready message.

```
# dmesg

… gadget: Gadget Serial v2.4

… gadget: g_serial ready

… g_serial gadget: high-speed config #2: CDC ACM config

# ls –al /dev/ttyGS*

crw-rw---- 1 root dialout 248, 0 Dec 31 1999 /dev/ttyGS0
```

Connect the Micro-USB cable between the host PC and the ODROID, which should give the following message on the host:

```
$ dmesg

… usb 1-1.2.4: new high-speed USB device number 3 using ehci_hcd

… cdc_acm 1-1.2.4:2.0: This device cannot do calls on its own. It is not a modem.

… cdc_acm 1-1.2.4:2.0: ttyACM0: USB ACM device

$

$ ls –al /dev/ttyACM*

crw-rw---- 1 root dialout 166, 0 Jan 22 00:29 /dev/ttyACM0
```

Data should now be transmitted through `/dev/ttyACM0` (PC), and `/dev/ttyGS0` (ODROID):

```
$ sudo chmod 666 /dev/tty-ACM0

$ echo qwerty > /dev/ttyACM0

# cat /dev/ttyGS0

qwerty
```

Emulating a ethernet storage link:

Ethernet over USB

Load the g_ether module on the ODROID:

```
# sudo modprobe g_ether
```

Launch dmesg to get the Ethernet Function gadget driver ready message, and details about the ethernet connection.

```
# dmesg

… gadget: using random self ethernet address

… gadget: using random host ethernet address

… usb0: MAC e2:53:36:a5:8f:38

… usb0: HOST MAC 76:42:28:9e:9f:eb

… gadget: Ethernet Gadget, version: Memorial Day 2008

… gadget: g_ether ready

… g_ether gadget: high-speed config #1: CDC Ethernet (EEM)

# ifconfig

usb0 Link encap:Ethernet HWaddr e2:53:36:a5:8f:38

inet6 addr: fe80::e053:36ff:fea5:8f38/64 Scope:Link

UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1

RX packets:106 errors:0 dropped:0 overruns:0 frame:0

TX packets:82 errors:0 dropped:0 overruns:0 carrier:0

collisions:0 txqueuelen:1000

RX bytes:22645 (22.6 KB) TX bytes:18987 (18.9 KB)
```

Connect the Micro-USB cable between the host PC and the ODROID and which should show the following messages on the host after typing "dmesg":

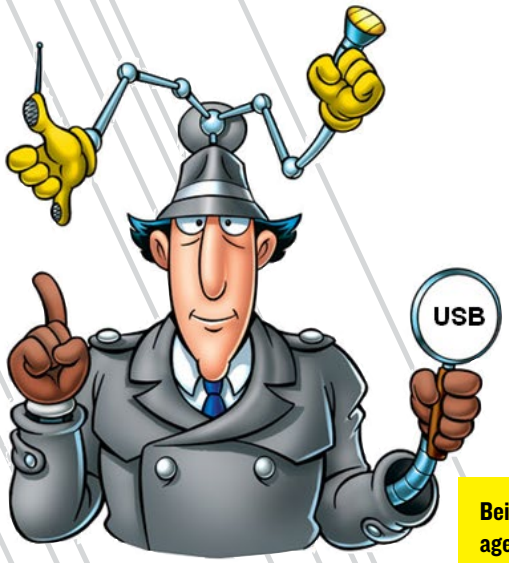
```
$ dmesg

… usb 1-1.2.4: new high-speed USB device number 12 using ehci_hcd

… hub 1-1.2:1.0: unable to enumerate USB device on port 4

… usb 1-1.2.4: new high-speed USB device number 13 using ehci_hcd

… cdc_eem 1-1.2.4:1.0: usb0: register 'cdc_eem' at usb-0000:00:1a.0-1.2.4, CDC EEM Device, 62:19:ce:95:e0:9f

$ ifconfig

usb0 Link encap:Ethernet HWaddr 62:19:ce:95:e0:9f

inet6 addr: fe80::6019:ceff:fe95:e09f/64 Scope:Link

UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1

RX packets:3 errors:0 dropped:0 overruns:0 frame:0

TX packets:4 errors:0 dropped:0 overruns:0 carrier:0

collisions:0 txqueuelen:1000

RX bytes:488 (488.0 B) TX bytes:624 (624.0 B)

$ sudo ifconfig usb0 192.168.100.1

$ ifconfig

usb0 Link encap:Ethernet HWaddr 62:19:ce:95:e0:9f

inet addr:192.168.100.1 Bcast:192.168.100.255 Mask:255.255.255.0

inet6 addr: fe80::6019:ceff:fe95:e09f/64 Scope:Link

UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1

RX packets:21 errors:0 dropped:0 overruns:0 frame:0

TX packets:33 errors:0 dropped:0 overruns:0 carrier:0

collisions:0 txqueuelen:1000

RX bytes:4677 (4.6 KB) TX bytes:8394 (8.3 KB)
```

Now the ODROID will be able to send and receive data through the usb0 port:

```
# sudo ifconfig usb0 192.168.100.2

# ifconfig

usb0 Link encap:Ethernet HWaddr e2:53:36:a5:8f:38

inet addr:192.168.100.2 Bcast:192.168.100.255 Mask:255.255.255.0

inet6 addr: fe80::e053:36ff:fea5:8f38/64 Scope:Link

UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1

RX packets:594 errors:0 dropped:0 overruns:0 frame:0

TX packets:434 errors:0 dropped:0 overruns:0 carrier:0

collisions:0 txqueuelen:1000

RX bytes:120002 (120.0 KB) TX bytes:98616 (98.6 KB)
```

Ethernet over USB

Ser a over USB



Storage over USB

Being able to work alongside network, storage and serial, will save your life more often than you think.

```
# ping 192.168.100.1

PING            192.168.100.1
(192.168.100.1) 56(84) bytes
of data.

64 bytes from 192.168.100.1:
icmp_seq=1 ttl=64 time=0.348
ms

64 bytes from 192.168.100.1:
icmp_seq=2 ttl=64 time=0.254
ms

^C

--- 192.168.100.1 ping sta-
tistics ---

2 packets transmitted, 2 re-
ceived, 0% packet loss, time
999ms

rtt     min/avg/max/mdev     =
0.254/0.301/0.348/0.047 ms

$ iperf -s

------------------------------
------------------------------
----

Server listening on TCP port
5001

TCP window size: 85.3 KByte
(default)

------------------------------
------------------------------
----

[    4] local 192.168.100.1
port   5001   connected   with
192.168.100.2 port 34764

[ ID] Interval       Transfer
Bandwidth

[    4]   0.0-10.0 sec     193
MBytes    162 Mbits/sec

# iperf -c 192.168.100.1

------------------------------
------------------------------
----

Client     connecting     to
192.168.100.1, TCP port 5001

TCP window size: 20.7 KByte
(default)

------------------------------
------------------------------
----

[    3] local 192.168.100.2
port   34764   connected   with
192.168.100.1 port 5001

[ ID] Interval       Transfer
Bandwidth

[    3]   0.0-10.0 sec     193
MBytes    162 Mbits/sec
```

Wow! 162Mbps is faster than normal 100Mbps Ethernet!

Emulating a multi function link:

The full package

A multi-function link emulates a mass storage, serial and ethernet connection using only one combo g_multi module:

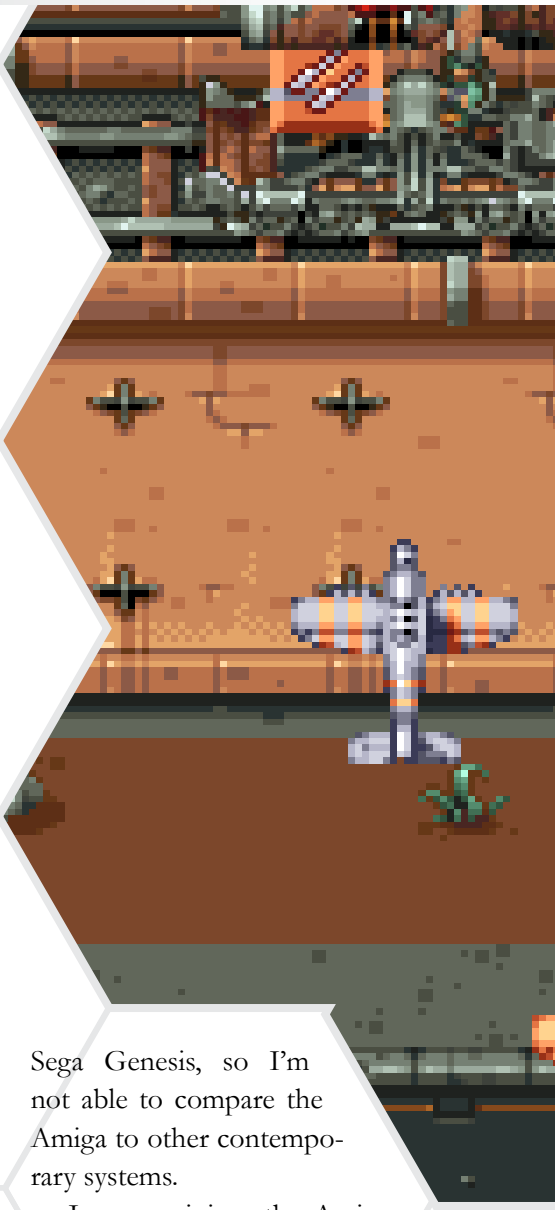
```
# mount

/dev/mmcblk0p1 on /media/boot
type vfat (rw,nosuid,nodev,fl
ush,umask=000)

# sudo umount /dev/mmcblk0p1

# sudo modprobe g_multi file=/
dev/mmcblk0p1
```

After connecting the ODROID's Micro-USB cable to the PC, all three functions listed above may be used simultaneously.

LINUX GAMING ON ODROID

THE RIGHT SYSTEM FOR YOUR GAMES

Tobias Schaaf



When it comes to gaming, everyone has different tastes. There are many gaming genres, including First Person Shooter, Role Playing, Arcade, Adventure, Simulation, Real Time Strategy, and many more. Some games are better suited for consoles, while others are only playable if you have a mouse and keyboard available. Since the ODROID platform can run many types of games, choosing your gaming system really depends on personal preference.

It also depends on whether you like to play on your own (single-player mode), with others (team or competition mode), on the internet (network mode), or with a friend in the same room in front of your TV (multi-player or split screen mode). It also depends on whether you want the old classics with the original beeping sounds, or prefer high fidelity audio and video with orchestral soundtracks and stunning 3D graphics.

This month, I give an overview of the gaming opportunities that the ODROID platform offers, giving you some ideas regarding what games you might like to explore.

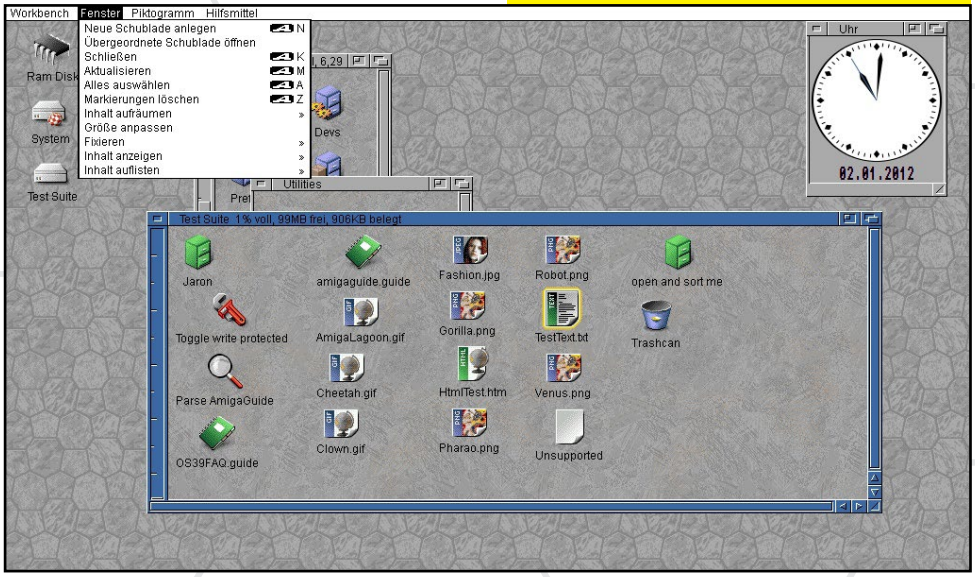
Finding the right system

Since the ODROID offers so many emulators and games, it's hard to figure out what system to choose from. So, I want to examine some of the gaming systems that the ODROID is able to emulate, and compare their pro and cons are to help you decide which is the best choice for you. This will somewhat reflect my own opinion, and might differ from other people's opinion, so please consider this article more as a suggestion than hard facts.

If you have already downloaded and installed my ODROID GameStation Turbo image, available for free on the ODROID forums, you have a good idea of what systems the ODROID is able to emulate.

First on the list is the Amiga. I had a few Amigas back in the day, and really loved to play games on it. In the 1990s, the Amiga was very popular. Compared to other PCs of the day (Apple //GS, Color Macintosh and IBM PC), it had improved graphics and far better sounds than most of the PC ports of that time. I never had an actual console like a Super Nintendo Entertainment System or a Sega Genesis, so I'm not able to compare the Amiga to other contemporary systems.

In my opinion, the Amiga was far ahead of its time. With games that took 15 disks or more, multi-



The Amiga desktop. This was the system that made all of your friends jelly, and you feeling like the coolest kid in school. No kidding!

actually has some games that can be played together with a friend, so I consider it a great multiplayer system. Games like Banshee or The Chaos Engine are really awesome to play with a friend.

Left: Banshee, a great Amiga shooter
Right: The Chaos engine

The SNES was a big improvement compared to the NES and other consoles, but that is not a big surprise if you compare the specs of the two consoles:
The NES had only about 2KB of RAM and 2KB of Video RAM, and a maximum of 25 colors simultaneously (out of 48 colors and some grays).
Its successor, the SNES, could use up to 15bit colors (32,768 possible colors), and had 128 KB of RAM and a large amount of Video RAM (64 kB main RAM, 512 + 32 bytes sprite RAM, 256 Đ 15 bits palette RAM).
The ROM size of games from the NES were between 8 KB to 1MB
On the SNES, the ROM size was between 256KB and 6MB.

tasking, the ability to use hard drives to store games and programs, and the Workbench Graphical User Interface, it was very impressive.

There are some emulators that run Amiga games well, but it takes lots of configuration to get the games to run, and often you have to switch between Kickstart disks (the BIOS of the Amiga). Simply switching through the many disks can also get somewhat annoying. Although the graphics of the Amiga was outstanding it can hardly compare to other consoles now.

I use retroarch in my ODROID GameStation Turbo image to run Amiga games, but unfortunately, not all of them run in full speed. So, I consider the Amiga an unstable system to actually use on the ODROID. There are some emulators that can run directly (for example, E-UAE and FS-UAE), which give better performance than the retroach port, but they require advanced configuration, and won't run easily with gamepads, which the retroarch port does well. There are some beautiful games on the Amiga, but most of them have been ported to other consoles or to the PC, which can be emulated more successfully.

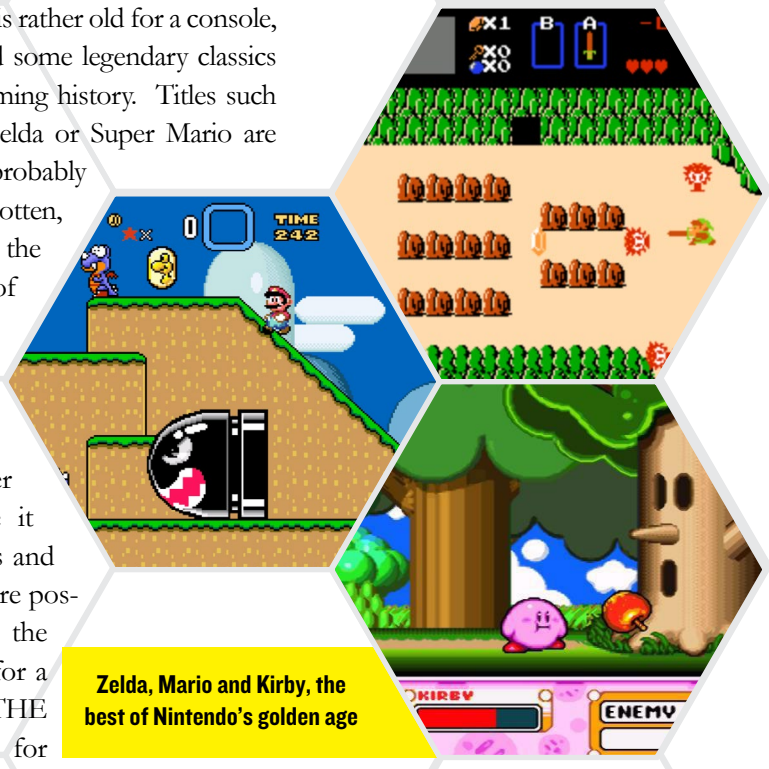
The benefit of the Amiga is that it

Next on the list of emulators is the Nintendo Entertainment System (NES), the Super Nintendo Entertainment System (SNES) and the GameBoy Advance (GBA). If people talk about Nintendo, and especially about retro games, you are going to hear about NES and SNES consoles. The NES is rather old for a console, but it introduced some legendary classics that changed gaming history. Titles such as Legend of Zelda or Super Mario are games that will probably never be forgotten, and are one of the most important of Nintendo's franchises.

The SNES was a great improvement over the NES, since it had more colors and offered a lot more possibilities than the NES, and was, for a very long time, THE gaming console for people, including professional gamers.

It was possible to pack much more data on a SNES cartridge, which allows for more content, and made games look a lot better, as seen in a direct comparison between the NES and the SNES:

The SNES also used video rendering layers, which allowed more effects and content to put in a single scene, and had a higher resolution than its predecessor.



Zelda, Mario and Kirby, the best of Nintendo's golden age

Another advantage of these consoles was that they were made for two controllers, which resulted in a large library of games that could be played competitively or cooperatively with friends. Between all of the consoles covered in this article, I consider the SNES to be the best console for multiplayer games.

In my opinion, GBA is often overlooked when it comes to retro gaming and Nintendo. The GBA was a handheld console from Nintendo, very much like the GameBoy (GB) and GameBoy Color (GBC), and was actually compatible with both GB and GBC games. It even had a second processor specifically for backwards compatibility, so that GameBoy owners could still play their older games on the newer console. Otherwise, it was very much like the SNES, which made the small handheld device very powerful for its time.

The GBA also had 15-bit color, and the CPU was 32-bit, as compared to the 16-bit processor of the SNES.
It also offered 32KB + 96KB of Video RAM (internal to the CPU) and 256 KB WRAM (outside the CPU).

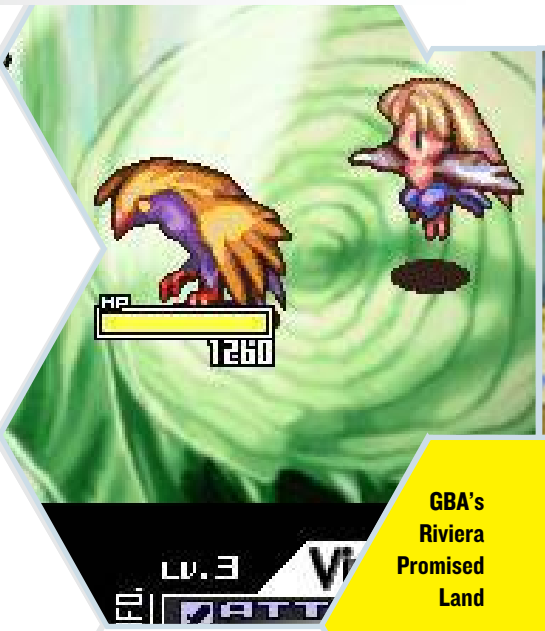
If you compare these specs, the GBA was a better system than the SNES in nearly every aspect. The only advantage that the SNES had over the GBA was the resolution, which was only 240x160 pixels. Even with the lower resolution, the cartridge size went from 1MB up to 32MB.

More size equals more content and once again, the GBA was a big improvement when it came to content. It offered movie cut-scenes, beautiful music, and very nice graphics, which is why I prefer GBA emulation over SNES. Many games that existed on the SNES were ported to the GBA, and they not only look much better, but in many cases, have additional content.

Although the GBA is, in my opinion, a far better console than the SNES, it has a downside. The GBA is a handheld console, and although it had multiplayer capabilities between two consoles via a connection cable, it makes it impossible to play GBA games with a friend on the same machine.

However, if you prefer single-player mode, you should definitely take a look at GBA games. You can also see if your favorite SNES games exist for GBA, and maybe get a newer, better experience from your game. If you're looking for a new game to try, you should definitely check out the GBA game library.

Overall, I really like the GBA, and I routinely play some GBA, since they have very attractive graphics and immersive gameplay. All of the Nintendo systems (GB, GBC, GBA, NES, SNES) run fine on the ODROID using the emulators provided on ODROID GameStation Turbo.



GBA's Riviera Promised Land

I prefer to run SNES in retroarch and play the other Nintendo emulators using mednafen since mednafen performs better than retroarch for these platforms, and allows software scalers such as hq2x or super2xsai, which can enhance the picture quality and make the games look more "modern" than they really are.

If you are a fan of retro Arcade machines, you'll probably enjoy MAME, NeoGeo and NeoGeo Pocket, which are available for the ODROID. MAME and NeoGeo are available on retroarch, and NeoGeo Pocket is available on mednafen.

NeoGeo Pocket is similar to the GBA, and is somewhere between a GBC and GBA, but it was not very popular, and lacks the large game library of its competitors. Still, it's similar to the other Arcade machines, and offers some challenging action games.

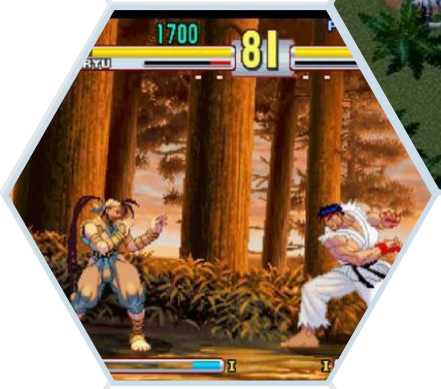
MAME and NeoGeo have a lot of fast action games like Street Fighter 3

Final Fantasy IV SNES vs GBA side by side.

GBA's Summon Night Swordcraft

3rd Strike and 1944 The Loop Master.

Arcade games vary from those that have simple graphics, to 3D games, to some with sprite technology, like the previous mentioned 1944 The Loop Master. If you like a lot of action and expert gameplay, the Arcade games that the ODROID can run is what you are looking for, especially when playing with friends.



However, the Arcade genre is limited, and implies fast action games. You will rarely find the RPG, Adventures or Simulation genres on an Arcade machine. It's hard to compare Arcade games with a SNES or Sega, since the Arcade games evolved a lot, and so did the hardware that was supporting these games. Some Arcade games have NES-quality graphics, while others have resolution as good as a PlayStation.

The ODROID platform can run most of these games without issue, and for this, it's really nice to have. There even is a project documented on the forum where someone built their own Arcade Machine at home with a U2: `http://bit.ly/1cmrgjK`.

Another big company back in the earlier days of console gaming was Sega, which directly competed with Nintendo. The Sega team had very good ideas, but in the end, upset a lot of developers and publishers, and now Sega only produces games for other consoles.

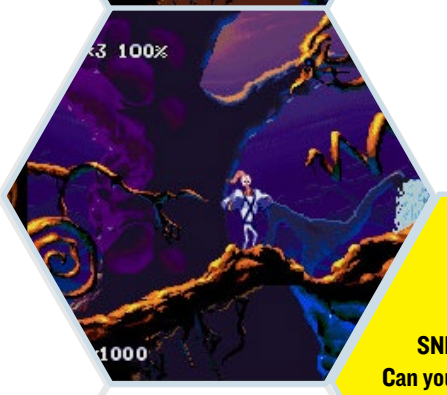
The Sega MasterSystem, Sega Genesis (Mega Drive), and the Sega GameGear are the emulators that I included in the ODROID GameStation Turbo image, and are probably their most well-known consoles. They also produced a couple more systems like the Sega 32X, Sega Mega-CD, Sega Saturn and the Sega Dreamcast, which was actually a really nice console, but it's hard to find emulators to run these Systems on Linux and ARM so they are not included in the image (yet).

Street Fighter 3 3rd Strike
1944 The Loop Master (NeoGeo)
Sonic the Hedgehog
Alex Kidd in Miracle World (Sega Master System)
Double Dragon (NES)
Double Dragon (SMS)

If you want to compare the different Sega systems to others of its time, you could say the Sega Master System is comparable to the NES, the Sega Genesis is similar to the SNES, and the Sega GameGear was also a Handheld like the GBA or GBC.

Sega Systems always tried to be just a little bit better than the Nintendo consoles. The Sega Master System had more RAM and more colors than the NES, and was slightly faster. The most popular game was not actually Sonic the Hedgehog, but Alex Kidd in Miracle World. Still, Sonic was Sega's answer to Mario and it worked well for Sega.

If you compare the games available for the NES and Sega Master System side-by-side, you can see that the Sega actually looked a lot better than the NES. If your favorite game on the NES is also available for the Sega Master System, I recommend playing the Sega Master System version instead.



A comparison of SNES and Genesis graphics. Can you tell which system is which? Earthworm Jim 2 Aladdin SNES vs. Aladdin Mortal Kombat

Sega tried to reach older gamers as well, and wanted to appeal to professional gamers. For this, the uncensored version of Mortal Kombat was published on the Genesis, which had blood and other effects that had been removed from the SNES version. Genesis also published many sports games that were very popular among adults.

Overall, the SNES had better hardware, and games looked more impressive than the Sega Genesis, but Genesis tried to counter that with better game mechanics and more serious games and gameplay.

The next generation, called the Sega Genesis, wasn't as good as the SNES, since it had less colors to choose from, and developers had a hard time porting games to the Genesis since Nintendo did not allow its companies to produce for any console other than Nintendo. Still, there were quite some nice games for the Genesis, and they look great.

Although you can see more details in the SNES version of the game, the Sega Genesis had better game mechanics and included a sword, which was not in the SNES version. Interestingly, Disney illustrators were involved in creating the animations in the Sega Genesis version, which made them look much better.

The Sega GameGear was meant to be a superior handheld device and the direct competitor of the GameBoy (GB), but it was far more than that. In fact, the Sega GameGear was even more powerful than the Sega Master System, and had an adapter to watch TV.

The Sega GameGear is somewhere in between the Sega Master System and the Sega Genesis. It actually had even more colors than the Genesis, but the downside of the GameGear is that the resolution was rather low even for its time, but that wasn't important for a handheld console. There were about 300 games available for the GameGear, and if it ran on the Sega Master System, there is a good chance that it was ported for the GameGear as well. Even some of the Genesis games were published for the GameGear, so give it a try and see what you like best.

The next console on the market to make an impact on gaming was Sony with its PlayStation, although by now it's considered retro as well. The PlayStation has an advantage over the previously mentioned consoles, because it's from a newer generation, but still it offers some console game classic that were updated for the new platform.

Looking at the quality of the PS1 games, there are beautiful movies, narrative speech, and high resolution graphics. The PS1 outranks all the previous named consoles, and if you find a game that was originally made for the SNES or Sega Genesis, you can expect the PS1 version to have lots of additional content like movie cut-scenes and probably better music.

Although the retroarch port of PS1 uses a software renderer to display its graphics, they perform fluently on the ODROID. Classics like Final Fantasy 7, 8 and 9 offer a few examples of its high resolution graphics. Even better, the PS1 was designed for multiplayer, which means you can play with some friends on the same TV.

Finally, we reach the most impressive emulator on the ODROID so far: PPSSPP, which can play PlayStation Portable (PSP) games on the

ODROID. The PSP runs a large collection of games that offer beautiful 3D graphics with movie cutscenes and real voice acting.

Although PPSSPP runs best on a PC running Windows, Linux or Android, I also successfully ported it to the ODROID. There are lots of PSPgames that run quite nicely on the ODROID in an X11 environment using PPSSPP.

The Sony age, PS One's Final Fantasy VIII, and the PSP games of Little Big planet, Soul Calibur, Fifa Soccer 2011, Wipeout Pure and Naruto Shippuden are not far from your ODROID

WHAT EMULATOR AND WHAT SYSTEM IS BEST FOR YOU?

If we compare the different systems by their power, look and feel, considering color depth, game content, music fidelity, and hardware technology, line up nicely from least powerful to most powerful:

GB < GBC < NES < SMS < GAMEGEAR < GENESIS < SNES < GBA < PSI < PSP

Not included in this ranking is the Amiga, which is more like an old time PC, despite its variety of games and other options. Amiga games ranged from very simple to spectacular, offers beautiful music and even more colors than an SNES or GBA - up to 24bit.

Also missing are the Arcade machines: NeoGeo Pocket, NeoGeo and MAME, since these systems offer a variety of games from the classic Pac-Man up to Tekken 6 for the PS1. Of these consoles, I prefer NeoGeo, since its games look the best and have an awesome look and feel, in my opinion.

Another way of deciding what games to try, might be whether the emulators offers the opportunity to play with other people together in front of your TV. For multiplayer gaming, the best consoles are MAME, NeoGeo, SNES and the PS1. There are also multiplayer games for Sega Master System and Sega Genesis as well, but I encountered them rather rarely, so I think that SNES would be the best choice if you want to play with a friend.

Although handheld devices such as the GBA might be the best single-player gaming console, they normally lack the ability to play together with someone else, unless you connect with a friend over the Internet or through your home network using two computers.

Each system that I covered in this article has its own charm. They all have some unique and engaging games, and it's usually the gameplay, not the graphics, that makes a particular game so enjoyable. With an ODROID, you have the possibility of experiencing all of them just by switching from one emulator to another. And, with ODROID GameStation Turbo, you have all of these emulators already set up, ready to play games with your favorite controller.

ESTIMATING RADIO NETWORK INTERFERENCE

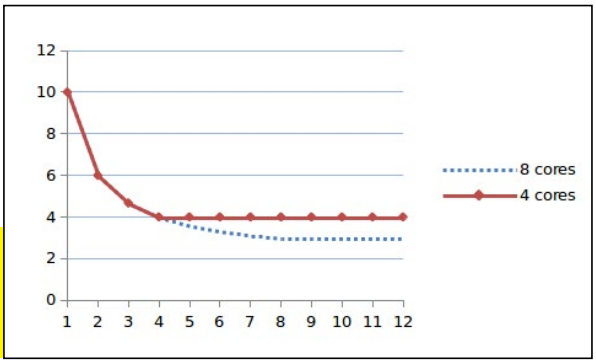
WITH MULTI-THREADED JAVA

Jussi Opas

The project described in this article involves the estimation of radio network interference using a multi-threaded Java application. How does the ODROID compare to other computers performing the same work?

If some tasks can be parallelized, then each additional thread shortens the execution time of the whole job. Given that there is an execution process available for performing each added thread, let's say that it takes 8 units to do some work sequentially. Adding one new thread shortens the time to 4 units, while adding a third thread further shortens to 2.67 units, and the 4th additional thread lessens the time to 2 units. If the work is not fully parallelizable, there is always some portion that must be done in sequence by one thread, and that part of the work cannot be accelerated by adding threads.

Let's start with the assumption that the non-parallelizable work takes 2 units to perform, and assume that adding more threads does not incur any extra cost. Then, we add up to 12 threads. The whole work should then be done in constant time from the 4th to the 12th thread. With this information, we can draft an expectation model, as shown by the square dotted solid line to the right.

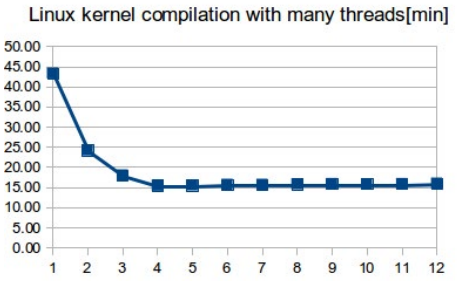


Expectation model; theoretical performance curves for multi core processors.

It is of interest to see that adding even more equally performing cores shortens the used time, but not by much, as shown by the dashed line. The sequentially done 2-unit work throttles the overall performance. If the full benefit of all cores is the goal, then also the previously sequential work should be parallelized. The compilation time of Linux kernel on ODROID XU demonstrates this expectation model well at right.

Our sample application, called Poiju, computes estimation of radio network interference at the beginning of next page. To do the estimation, combined coverage and service area of each radio cell must be calculated. Then, at each pixel, the field strength of interfering cells is computed. This is done over the service area of carrier cells. Computation is made on pixels of size from 30 to 350 meters. The size of radio cells may vary from 200 meter to 50 kilometers.

The computational challenge is exponential in its nature, as the amount of pixels to visit and to compute grows in potency when a more accurate resolution is used.

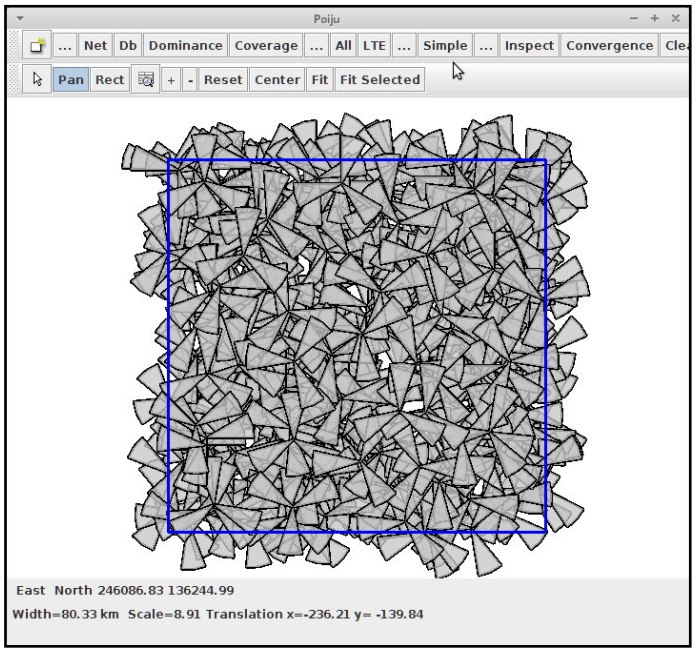


Compilation time of Linux kernel on XU, the time unit in minutes.

The characteristics of the program are that it uses only RAM (no file access), and all work can be done by the CPU, since no GPU-related work is performed. Parallelization has been designed so that, for each cell, service area is computed first sequentially, and after that interference is computed with many threads for cell specific closest candidates.

Physical cores and/or hyper threads are visible as available processors [LEA99]. Java threads hide underlying operating system and hardware, so the same program can be used on all platforms whereever a virtual machine is available. To implement threads, we used Java's modern concurrency package.

A typical development session with the application Poiju is shown on the next page. The prediction parameters dialog is shown in the front, and the main window of the Poiju ap-



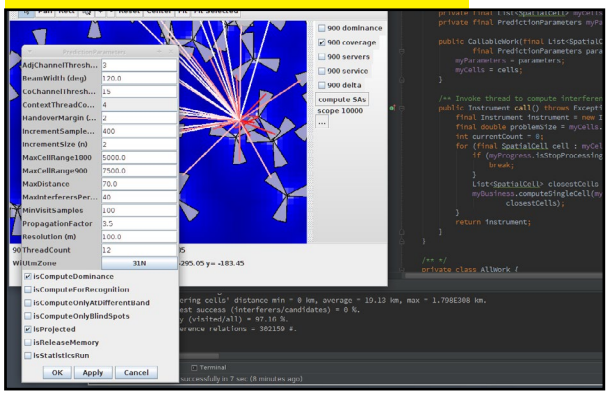
plication is below it. Also shown are the htop terminal and the lowest application is the development environment called IltelliJ IDEA.

The ODROID-XU's ARM processor offers 4 power efficient A7 cores and 4 high performance A15 cores [ARM13]. At any one time, either a cluster of little cores or a cluster of big cores is working, known as big.LITTLE architecture. Hence, there are 4 processes available. The Poiju application is using only the big cores while intensive computations are made.

The performance of application is measured with self made instrumentation by taking core-specific times prior to doing something, and then the next time right after doing the task:

We are doing all tests with 10,000 radio cells at each time. The performance

Poiju in development session on ODROID XU on a monitor via display port.



```
        final long t1 = System.
nanoTime();

        final double distance =
SpatialMath.computeProjectedDi
stance(carrier, candidate);

                        instrument.
incrementDistanceTime(System.
nanoTime() - t1);
```

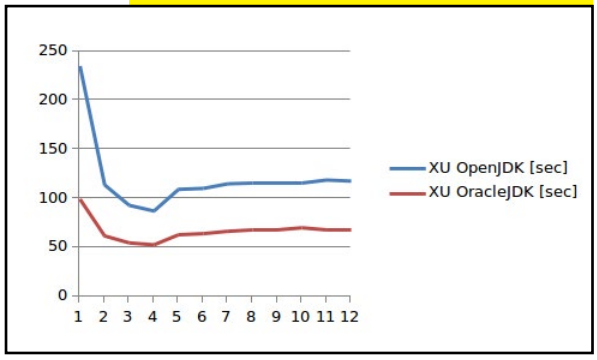
measurement results with OpenJDK and OracleJDK are shown below right.

The first observation is that our theoretical model, as presented before, is being partially followed. Each added thread is speeding up the program execution, until the 4th thread. The first added thread yields the best performance boost. This is partially affected also by Just In Time (JIT) compilation. JIT improves execution performance when same byte code is executed repeatedly.

Adding the 3rd and the 4th threads predictably improves performance, while the addition of the 5th thread decreases performance. We guess here that thread scheduling and context switching does not come for free, when there are many threads waiting for execution time. When we know this, it is better to limit the number of threads to 4 in the application program. That can be done by calling the Runtime.getRuntime().availableProcessors() method. We see also that OracleJDK is performing better than OpenJDK. OpenJDK is the default installed Java on Ubuntu, so the user must manually install Oracle JDK. Fortunately, it is freely available from the Oracle Developer website.

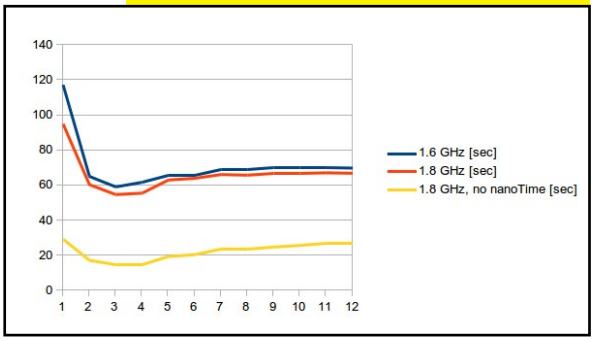
The performance of the application on XU is practically identical between 4 different Linux distributions, Debian, Xubuntu, Fedora and Ubuntu with Mate desktop. There is a small, but hardly visible, improvement in JDK releases 1.6, 1.7, and 1.8. It is also of interest to see how performance is improved when frequency is changed from 1.6 to 1.8 GHz, as shown in at end of this page.

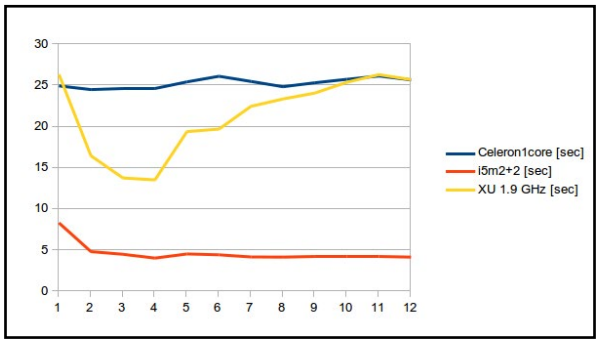
Running times on 4 A15 core XU board with two different JDKs.



With htop and vmstat we see that more than half of the time is used by kernel instead of user space. This is a major problem that we so far noticed only on XU. What is wrong in the application? If there is some internal error in a program and a lot of exception are thrown and treated silently, it would affect the performance significantly. However, that should not be visible as kernel time. After several other similar

Over-clocking improves performance, but dropping usage of self-made instrumentation improves it significantly.



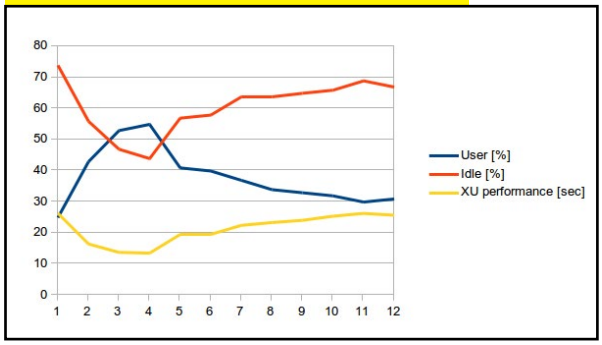


guesses and trials we were not able to find the reason. Finally, we commented self-made detailed instrumentation that was being generated with Java's nanoTime method. As a result, two-thirds of the computation time disappeared.

We can expect that nanoTime inefficiency and any other similar feature of Java on ARM will be streamlined in future, as there is news of co-operation [JAC13]. Since we let the ARM processor do the work that traditionally has been assigned to desktop computers, we also made a comparison with two laptop computers, one having a Celeron processor at 2GHz and another having an i5-2520M at 2.5GHz. The Celeron has one core while the i5m has 2 cores with 2 hyper threads. We refer to these processors as x64 to signify that they are 64 bit x86 processors. The comparison is shown above.

With this different scale in the graph, it is obvious that something starts to throttle the performance from the 5th thread on. Adding more threads, up to 12, does not seem to degrade the performance of the one core Celeron at all. Meanwhile, we see that the application performance at XU is being degraded when 5 and more threads are used. From the vmstat output we can produce a graph, where user space time and idle time is shown together with performance, see the figure at bottom left.

Processor is not loaded, instead there is free capacity.

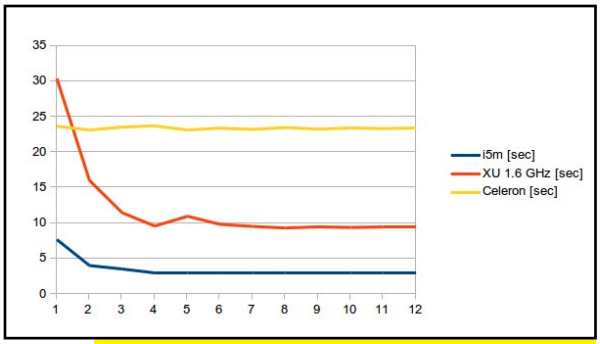


We don't know the actual reason for the behavior, but we can guess things like context switching is not efficient, or work load is not even but unbalanced, or the share of sequential work is large, or something similar. The first guess can be sorted out with the following reasoning: 1) there are no problems in multi-threaded Linux kernel compilation, 2) literature search shows that context switch time should be at maximum 0.4 % of overall time [DAV07], and 3) the development environment - IntelliJ IDEA - uses multiple threads flawlessly. We leave the cause as a puzzle to be solved.

For an alternative approach, we implemented threading differently, and also made a couple of other changes. Now, cells are organized into as many work lists as there are threads to be used. Both service areas and interferences are computed in parallel, so there are N cells being computed at same time. The work is now fully, or almost fully, parallelized, and the performance of the application clearly improved. The Performance graph follows now our expectation model. Multi-threading uses the whole capacity of the XU board. In this comparison, the ODROID-XU was used at 1.6 GHz with the performance governor option.

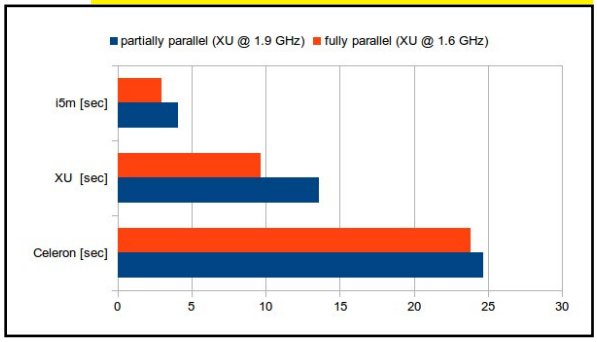
To make a more legible comparison, we take times at the 4th thread and put the numbers of partially and fully parallel runs into one illustration below.

In this test, the multi-core ARM processor is doing its jobs much faster than the one core Celeron processor. With 4 threads, the performance is 1.8x better. Meanwhile, the mobile i5 processor is 3.3x faster than the XU. Fully parallel thread configuration changes these numbers to 2.5x and 3.2x respectively. In this comparison, the XU does 1/3rd of the work that an I5 can do at the same time.



Performance profile when the application is fully parallelized.

Comparison of i5, XU and Celeron when 4 threads are used.



Summary

The ODROID XU board performs the same Java tasks for which desktop computers have been traditionally used. Although it can't compete against the brand new (and more expensive) x64 processors, it can be used as a replacement for a mobile i5. The ODROID-XU is clearly better than an older single-core x64 computer, and Java applications benefit signifi-

cantly from the usage of multiple threads.

The default on-demand governor of Linux is biasing performance test results, because the CPU's clock frequency is being changed automatically. Therefore, in Linux, one should use performance frequency scaling governor during performance tests.

The System.nanoTime method costs a lot of kernel time. The nanoTime method problem affects both Celeron and ARM processors, while the mobile i5 processor does not suffer from it as much. The call to nanoTime should be commented out in production code.

When high work load is given to parallel threads, then an equal amount of threads to the number of available processors should be used. The `Runtime.getRuntime().availableProcessors()` method can be can be used for this purpose.

Use Linux tools like vmstat and htop to validate loading and kernel time. For profiling an application, jvisualvm in JDK's bin directory can be used. Don't trust self-made instrumentation without a backup measurement tool.

Through this exercise, the performance of the Poiju on other platforms was improved by 40%. On the ODROID-XU alone, the improvement was huge from the initial design stage. In general, desktop Java applications can be improved when they are also tested on the ARM-based XU. The ODROID computes interference estimate to 10, 000 radio cells in 9.6 seconds by using pixel based computation and multi-threaded Java.

This exercise has been made as an ad-hoc, BYOD type project. The content here as such does not imply anything about actual products of NSN.

References

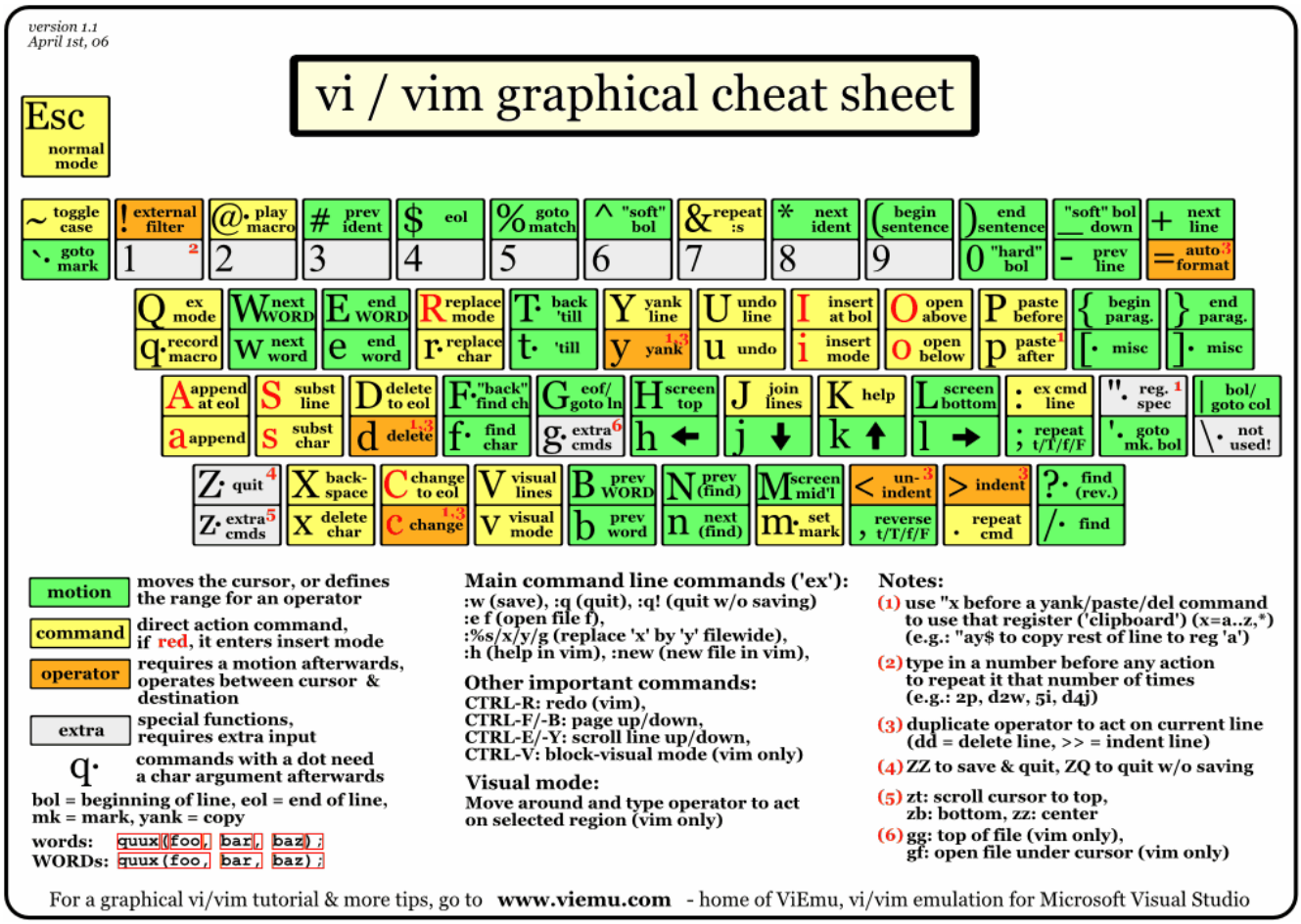
[DAV07] David F. M., Carlyle J. C., and Campbell R. H. Context Switch Overheads on Mobile Device Platforms. University of Illinois at Urbana-Champaign 201 N Goodwin Ave Urbana, IL. 5 pp. 2007. http://www.cs.huji.ac.il/~feit/exp/expcs07/papers/136.pdf

[LEA99] Lea Doug. Concurrent Programming in Java, Design Principles and Patterns, Second Edition. Addison Wesley. ISBN 0-201-31009-0. 411 pages. November 1999.

[ARM13] ARM Limited. Multi-threading technology and the challenges of meeting performance and power consumption demands for mobile applications. 9 pages, 2013.

[JAC13] Jackson Joab. Oracle and ARM to tweak Java. 2 pages. 2013. http://www.javaworld.com/article/2078833/enterprise-java/oracle-and-arm-to-tweak-java.html

Dear reader: I honestly have to do a better cheat sheet than this one, but this fellow one helped me on countless times, and believe me it will help you immensly for our next page article. Sincerely Bruno Douche



HOW TO INSTALL REBOL

A BEGINNER'S GUIDE

by Bohdan Lechnowsky, Editor

Rebol (Relative Expression Based Object Language)is a revolutionary advancement in programming language emerging from over thirty years of language research. It offers enormous flexibility and power, with a focus on intuitive language patterns that promote new ways of thinking about software tasks.

Follow these steps to install open-source Rebol 3 with GUI support:

Open a web browser and navigate to `http://development.saphirion.com/experimental/builds/android/`

Download `r3-droid.apk` **(amazingly, it's smaller than 2MB).**

When finished, double-click on the download icon (usually by the clock) and grant installation permissions.

Go to the apps list, and click the icon for R3/Droid.

Additional Android notes:

When using the Android Terminal app, it is almost always necessary to have root permissions. To login as root, enter "su" at the command prompt. This needs to be performed each time Terminal is opened.

On my installation of Android, I commonly get a message when I try to create a new file, stating "Read-only file system". To overcome this, enter "mount -o remount /" at the command prompt. This needs to be performed each time Terminal is opened.

To edit a script using Terminal, enter vi myscript.r where "myscript.r" is the name under which you want your script to be saved (see "vi cheat sheet" above for some basics on how to use it).

After editing your script, you will need to change the permissions so it can be accessed by Rebol 3 (example: `chmod 755 myscript.r`)

vi Cheat Sheet

"vi" was created back in the days when there were very few special keys on computer keyboards. Keys like "Esc" (escape) and the arrow cursor keys didn't even exist. One of the great advantages of the vi editor is that it is available on nearly any unix-based operating system, including Android.

Because of the early limitations of computer keyboards, vi was designed to have two modes. The first is the input mode, and the other is the command mode. On the Android version of vi, these two modes are toggled by pressing CTRL and [(more simply referred to as CTRL+[). During input mode, any key you type is entered at its face value. During command mode, here are some helpful commands:

```
i    Insert at current position
I    Insert at beginning of line
a    Append at current position
A    Append at end of line
dd    Delete current line
yy    Yank (copy) current line
.    Repeat last command
cw    Change current word
dw    Delete current word
p    Paste yanked data
:    Enter extended command mode.
Here are a few extended commands:
w    Write (save) the file
q    Quit (don't save) the file
wq    Write and quit
q!    Quit, even if the file has
been changed since last write
/    Search forward for the text
following the slash
?    Search backward for the text
following the slash
```

There are much more in-depth vi cheat sheets online, but the above commands should allow you to edit just about anything you need in vi. Although it may seem uncomfortable to use at first, seasoned vi users can often edit more efficiently than users on any other editor, graphical or not.

On Ubuntu/ARM

As open-source Rebol 3 is still in very active development, the graphical port of Rebol for ARM has not yet been released. According to the developers at Saphirion, they predict a working ARM port of Rebol 3 with graphics to be available sometime in February or March 2014. Keep checking the `http://development.saphirion.com/experimental/builds` directory

for an ARM port. The graphical version for Linux x86 is already available, but it won't work on your ODROID. The Linux ports are being helped heavily by a partnership with Saphirion and other commercial interests, so expect a lot of movement in this area over the next few months.

In the meantime, Rebol 3 is available in the non-GUI variety from `http://rebolsource.net`, and can work wonders as a replacement to bash, perl, PHP, Python and other languages that don't have a native GUI.

Follow these steps to install Rebol 3 on Ubuntu/ARM:

Open a browser and navigate to `http://rebolsource.net`.
Locate the download for Linux ARM. It comes in two variants: hard float and soft float. Either should work fine, but if you have trouble, the soft float version is likely more compatible with most systems.
After downloading, open Terminal and rename the downloaded file to "r3" for ease-of-use (example: `sudo mv r3-linux-arm-g4d9840f r3`**)**
Also, change permissions to allow executing the file (example: `sudo chmod 755 ./r3`**)**
To run it, enter sudo `./r3`**. To execute a script located in a file, enter sudo** `./r3 myscript.r`**. You can even execute scripts located on the Internet like this:** `sudo ./r3 http://mysite.com/myscript.r`.

If opening Rebol 3 without launching a script, you can now type any Rebol commands at the command prompt.

PROGRAMMING WITH REBOL

REDUCING COMPLEXITY IN DEVELOPMENT

by Nick Antonaccio and Bohdan Lechnowsky, Editor

Why Rebol?

Modern software development tools are overly complex. Creating even a small program to allow users to complete the most basic computing task typically requires the installation of a complex Integrated Desktop Environment (IDE), a bloated Software Development Kit (SDK), and other supporting tools. Developers need a working knowledge of diverse chains of tools patched together, along with library Application Programming Interfaces (API), database systems, and more just to create a simple application.

For most developers, the thought of building mobile apps, desktop GUI programs, and web applications represents mastering a tremendous variety of in-congruent work flow patterns, incompatible data formats, and inconsistent development methodologies that sap productivity at every turn. For these developers, it's an unavoidable part of the job.

We live in a time when ubiquitous and inexpensive handheld wireless devices and ODROIDs connect us all to an enormous world of useful data resources. Those devices enable critical data management, communications, business transactions, entertainment, and other practical benefits which deeply affect the nature and quality of our daily lives.

The primary frustrations of device ownership have been eliminated by new generations of technology. Users don't need to know how networks work to connect to the Internet. Cameras, microphones, speakers, GPS and other sensors are built into every tablet, phone, netbook, and desktop PC, and they work transparently without having to install hardware, drivers, or OS patches.

Tiny form factors, like ODROID, and inexpensive cellular network connections, enable new types of applications that weren't practical, even on laptop machines, a few years ago. The high-speed quad-core ODROID processors, not to mention even the least expensive modern phones and tablets, run circles around the best desktop PCs of a decade ago. Deep data content of every sort, in every field of study and interest, is available instantly online to anyone around the globe. Mainstream handheld hardware is tiny, the interfaces are beautiful, standard formats exist for virtually every type of data (images, audio, video, structured tables of text and numeric data, etc.), and well known operating systems and software all function in a relatively uniform and familiar way, so that everything just "works" the way that users expect.

Being able to use the amazing programmable network-connected computing power in our pockets, and all around us, makes available enormous cultural potential that has never before been available, and gives us the ability to create custom applications which leverage that tremendous power will only continue to grow in importance for future generations.

This situation is fantastic for users of "apps", but for software developers, the current landscape of tools is a painful mess. The remnants of every

legacy software and hardware solution still haunts and infects the process of writing modernized code. Numerous attempts to standardize on different commercially-motivated platforms and formats have led to the need to support a huge variety of data structures, language syntaxes, and tool sets.

To even begin creating a typical "hello world" app for Android, you need to install hundreds of megabytes of software onto a desktop computer, including an Integrated Development Environment (IDE), a Software Development Kit (SDK), a device emulator, and any "productivity enhancing" tools that help ease the wildly complex process of writing an applications.

Even for seasoned developers, modern application development is a weighty endeavor. If you also intend to port your applications across mobile, desktop and web platforms, you need truly deep experience, skill, and lots of time.

It's not uncommon to see teams of professional engineers devote themselves to building and supporting a single application. Despite the fact that Android is an open platform, today's devices feel closed, or at least cumbersome, for developers. The hurdles to learning how to program are too much for all but full time developers. Gone are the days when even hobbyists could write useful custom applications for their personal and business computing devices.

There was a time in our history when this activity was just as popular, accessible and even fun for users as running commercial apps. It's a shame, because never before has there been such a tremendous variety of knowledge and useful computing capability sitting at our fingertips.

Rebol On All Platforms

For more than a decade, a small group of developers, coding in Rebol, have known what it means to be truly productive, using a single simple tool to create applications of all types, for every popular platform that has come and gone, includingmore than 40 hardware platforms and operating systems so far, and the web.

All the core components that make modern computing possible - graphic user interfaces, network connectivity, manipulation of standard data formats, etc. are handled in Rebol using the simplest possible syntax, and implemented using a tiny interpreter that runs exactly the same way on every operating system. Rebol was designed from the ground up to eliminate complexity caused by patching together disparate tools, and it succeeds brilliantly in that regard.

Rebol side-steps most of the mess that has evolved over the years in mainstream software development technology, and for a wide range of common development tasks, it "just works" quickly, easily, and more simply than can be imagined by anyone mired in the mess of traditional programming tools.

Rebol has a special ability to wrap new capabilities into simple language structures (also known as "dialects", domain specific languages, or "DSL"s) which control multiple layers of computing functionality. It goes far beyond the abilities promised by old, rigid Object Oriented Programming (OOP) approaches, and it has a proven record of reducing complexity in widely varied development tasks.

Recently, Rebol version 3 ("Rebol 3" or "R3") was released by Carl Sassenrath as an open source project, and a version for Android named "Saphir" is forked and maintained by the Saphirion group. Saphirion is a commercial application development company that uses Rebol as their primary development tool. Free, open source R3 Saphir releases also exist for Windows, Mac, and Linux desktop operating systems, and R3 server versions enable easy and portable web development strategies.

In most cases, Saphir R3 download sizes range between 0.5 and 1.5 megabytes, with GUI, networking, and all other required components included, and it's shockingly simple to learn and use. An entire development toolkit for R3 requires less than a minute to install, since absolutely no hefty SDK or IDE installations are required, even for Android development.

Rebol works the exact same way on every platform, without any changes to code or workflow routine. Even people who spend the majority of their time doing things other than writing code can learn to create powerful custom business and personal applications with it, quickly and easily, on any chosen platform.

Despite its extremely simple nature, R3 is not a hobbyist toy or limited learning environment. Rebol is a powerful and deep professional tool which has proven itself in numerous critical commercial projects, across a wide array of demanding industry environments, around the world, for more than a decade.

Examples of Rebol's practical application in the real world

Quick schedule apps

Email programs with special features for personal use

Business applications to handle inventory, sales and employee management

Niche commercial applications intended for international re-sale

Distributed apps to organize group activities for your local school

Web apps to manage membership routines for an online club

Hardware interfaces to control computerized machinery in your home or business

Systems to manage robotics at brand name manufacturing facilities

Evolution of Rebol

For most of its life, Rebol 2 was a closed source commercial tool which

thrived as a "secret weapon" among only a small community of users who communicated privately and covertly using invite-only communication channels, the executables of which were developed using Rebol 2.

With the release of open source R3 and the new potential for Android development, in addition to more than 40 legacy platforms and web development, Rebol has attracted a new group of coders interested in its practical and productive capabilities for creating software of all sorts. In future articles, we will show you how easy it is to get started.

Red On All Platforms

During the closed-source era of Rebol 2 and 3, several languages were developed as open-source variants of Rebol including Boron, Topaz, World and Red. The data-interchange format, JavaScript Object Notation (JSON), was also heavily inspired by Rebol's clean data format.

The front-runner of these efforts presently is Red, which also has a low-level subsystem called Red/System. While Rebol is an interpreted-only language, Red and Red/System can run either as an interpreted language or as a compiled language. Red's founder and lead developer is Nenad Rakocevic, aka "DocKimbel".

The Rebol language family was recently recognized as the most expressive multi-purpose languages, as seen in the chart at `http://bit.ly/1iikG1r`. Expressiveness is the measure of what can be accomplished by a given amount of coding.

The only two languages rated as more expressive than Rebol are Augeas and Puppet, both of which are not general-purpose. Red falls in the same category as Rebol as far as expressiveness is concerned.

Red, along with Red/System, is the first real contender to be a "full-stack" programming solution. Several popular languages are compared and contrasted at `http://bit.ly/1tHdcbS` for their "Natural scope of application". Notice how Red, coupled with Red/System, cover the entire stack of application abstraction levels, even more so than Rebol itself.

Another language to learn? I've heard it all before!

Before starting on this section, make sure you have completed the Rebol 3 installation outlined in the article by Bohdan Lechnowsky in this same issue. The screenshots shown in this and future articles are taken from a Windows workstation, but will look nearly identical on Ubuntu, Android, or any of the other operating systems supported by Rebol 3.

For years, developers have been hearing that some "new" language technology will supposedly cut their development time by orders of magnitude. That sort of talk is so common that it falls on deaf ears. This introductory section provides a few short examples to substantiate some of the productive capabilities which Rebolers enjoy. Nothing else even comes close to its simplicity.

Using R3, you can build a GUI "hello world" application for Android, Linux, Windows, and Mac as simply as this:

```
load-gui

view [text "Hello World!"]
```

On Android, there will be no "close" button on the window, so press "Escape" on the keyboard to get back to the Rebol command line.

The program above is much more than the typical text-based console "hello world" app. It's actually a complete windowed form, capable of displaying all sorts of other useful Graphic User Interface (GUI) widgets.

Here's another simple program that displays some text entry fields, buttons, lists, and other common GUI widgets. Notice that every single widget word maps directly to something that appears on screen. There is no wasted syntax - you really don't even need to understand anything about "programming" to follow what this code does:

```
load-gui

view [

   field

   area

   check

   radio

   text-list

   text-table

   drop-down

   button

]
```

On Android, the Rebol command line currently only takes single-line commands. This is not a problem, as Rebol is syntax-free. Simply enter the command like this on Android:

```
load-gui

view [field area check radio
text-list text-table drop-
down button]
```

Alternatively, multi-line programs can be written with any text editor, and executed using Rebol. For instance, if you create a file called gui-demo.r that you can execute from the Rebol 3 command line:

```
do %gui-demo.r
```

That's it!

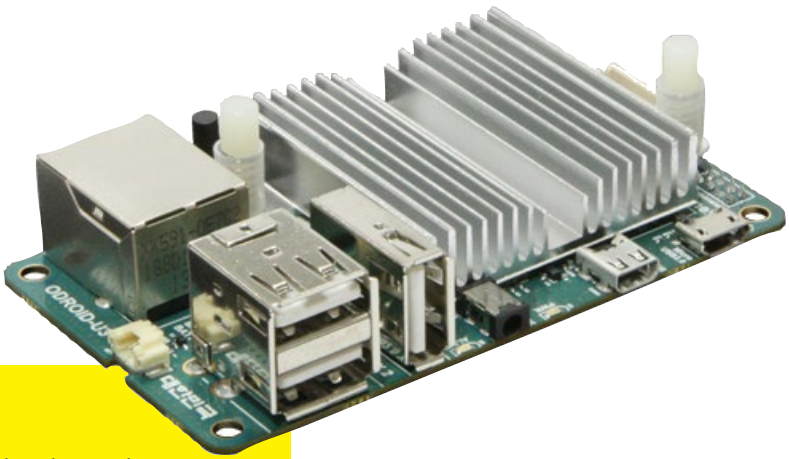
Pleas note that Rebol has numerous GUI dialects. The examples shown in this and future articles make use of the "R3-GUI" dialect, which is based on, and very similar to, the popular Rebol 2 Visual Interface Dialect (VID).

In the next article, we'll look at creating a fully functional text editor, graphical calculator, and more!

I/O SHIELD ACCESS

USING THE C/C++ LANGUAGE FOR ODROID-U3

Justin Lee and Kevin Kim



One of the exciting accessories available for the ODROID-U3 is the I/O shield, which was designed as versatile, general-purpose data acquisition and control module to provide a direct connection to the ODROID-U3 I/O connector.

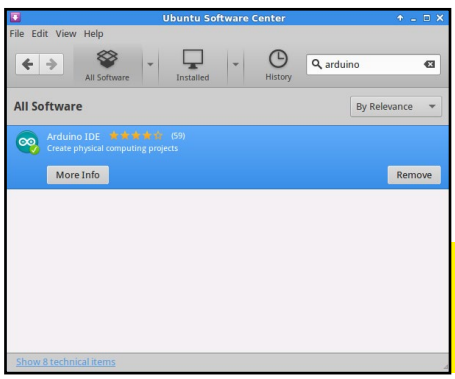
The I/O shield includes a big prototyping area, so you can wire up DIP chips, sensors, and more. Along the edges, all the GPIO/ADC/PWM and power connectors are broken out to 0.1" pins for ease of access. There are also two 3-pin headers for small servo motor connections, and a 10-pin connector of I2C/GPIO for further expansion.

This article will get you started with using the I/O shield using examples in the C++ language.

Installation

The drivers are installed by default on Ubuntu 13.10, which is available for download from the ODROID forums.

Verify that you have the following module (Ti's TCA6416A I2C to Parallel Port Expander) in your kernel source tree: `/lib/modules/`uname -r`/ kernel/drivers/gpio/gpio-pca953x.ko`



Launch Ubuntu Software Center, search for "arduino", and install it

Usage

Download the Firmata software for I/O shield, and setup the configuration as follows:

Port setup :
Tools > Serial Port > /dev/tty-ACM99

Open the example code and upload it to your IO_shield:
File > Examples > Firmata > StandardFirmata

Download the I/O shield source code `http://dn.odroid.com/U3_Accessory/u3_IOshield_example.tgz`

I. Extract source code using the "tar -xvzf" command.

2. Build the libfirmatac.a library.

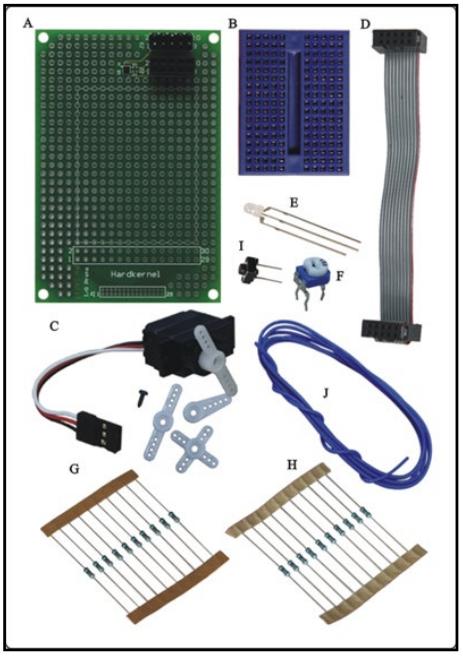
3. Build the Example code which links libfirmatac.a:

ledBlink.ex is an IO Shield 'D13' port LED blink example.

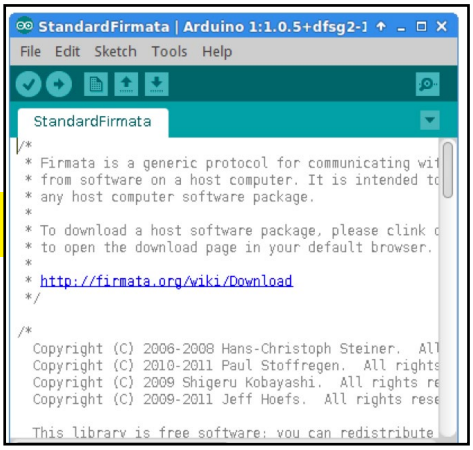
servotest.ex is an IO Shield 'D3' port servo motor control example.

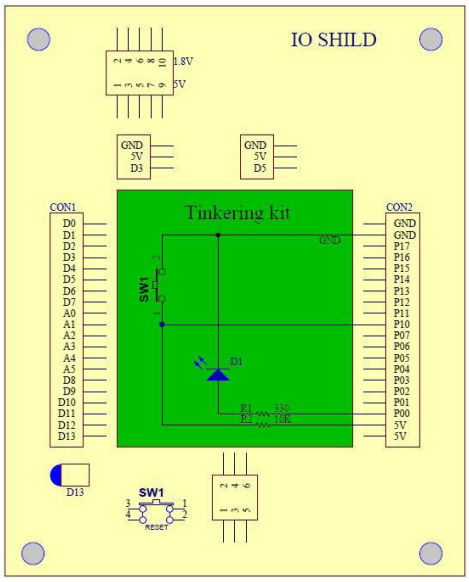
Making a Test Circuit with the U3 I/O shield and the Tinkering Kit

Use the following parts from the tinkering kit, also available from the Hardkernel website: B, C, E, G, H, I, J:

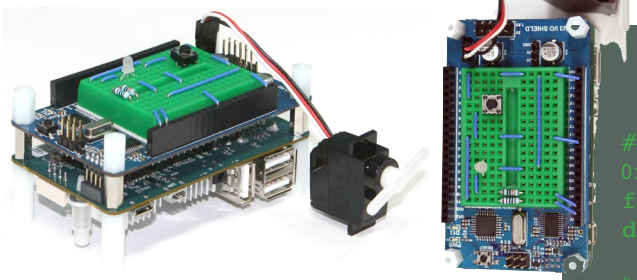


Firmata example code





Match the simple circuit shown here, and have your U3 like we are showing below:



Accessing the GPIO from the Linux User Space

GPIO means "General Purpose Input/Output", and is a special pin present in some chips that can be set as either input or output, and is used to move a signal high or low (in output mode) or to get the signal current status (in input mode). Usually, these pin are directly managed by kernel modules, but there is an easy way to manage these pins from the user space as well.

Standard Linux kernels contain a special interface which allows access to the GPIO pins. By running the kernel menuconfig command before compiling the kernel, you can easily verify if this interface is active, and enable it if necessary. The kernel tree path is `Device Drivers > GPIO Support > /sys/class/gpio/... (sysfs interface)`

If you embed the driver, you can access the GPIO via the kernel node `/sys/class/gpio`.

If you want to work with a particular GPIO, you must first reserve it, set the input/output direction, then start managing it. Once you reserve the GPIO and finish using it, you also need to free it, to allow other modules or processes to use them. This rule applies whether you want to use the GPIO from the kernel or at the user level.

GPIO Export

By typing "sudo modprobe" in the Terminal window after booting the Linux kernel, the GPIO driver and TCA6426 devices will be attached to the i2c-gpio.4 bus. 16 Ports are mapped to GPIO #289~#304 (I/O Shield pin number P00~P07, P10~P17):

```
# sudo modprobe gpio-pca953x

# sudo su –c 'echo tca6416 0x20 > /sys/devices/platform/i2c-gpio.4/i2c-4/new_device

# dmesg

… i2c i2c-4: new_device: Instantiated device tca6416 at 0x20
```

Next, build the u3_shield_GPIO_sysfs.c, located in the examples folder, using g++. Please note that the GPIOsys_init() function need the super-user permission to do modprobe and attach i2c device. The default password on the official Hardkernel images is "odroid".

`P00 port : LED blink`

`P10 port : key press, then GPIO 297 read value = 0, otherwise GPIO 297 read value = 1`

Starting your application software on boot

There's a place on the kernel to load modules as needed. However, you can't use it to send echo commands or values to sysfs without root or super-user permissions. Fortunately, there is a way to overcome this limitation by using a conf file to launch the module as a daemon, then using a shell script to perform the initialization:

1) Create the file `/etc/init/tca6416.conf`, which may require root privileges:

```
description "TCA6416 Module Initialization"

start on runlevel [2345]

exec /usr/sbin/tca6416init.sh
```

2) Create the file `/usr/sbin/tca6416init.sh`:

```
#!/bin/bash

# Load the Kernel Module

modprobe gpio-pca953x

# Set the I2C address of the module

echo tca6416 0x20 > /sys/devices/platform/i2c-gpio.4/i2c-4/new_device

# Enable all GPIO's on the board

for gpio_n in `seq 289 304`

do

echo $gpio_n > /sys/class/gpio/export

done
```

By adding these two files, all the GPIO ports will be permanently enabled on reboot. You can also add your application software in the "tca6416init.sh" as an alternative method.

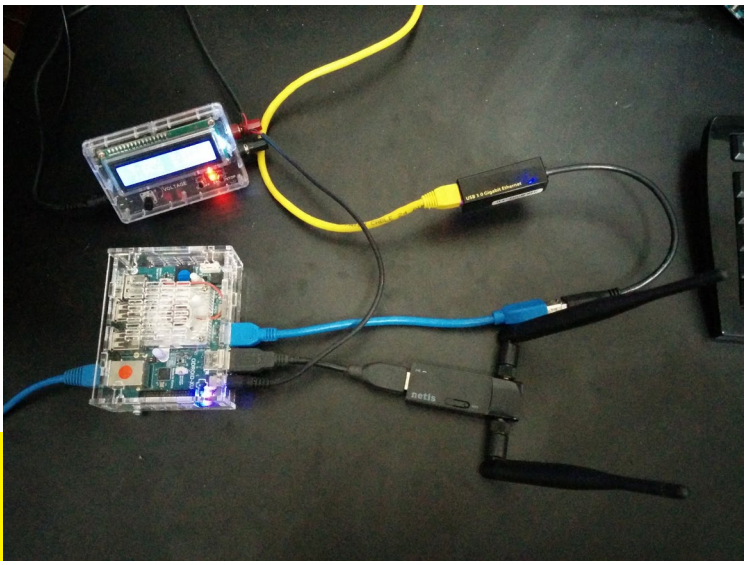
Additional Credits

The FirmataC library and examples are made by jdourlens, which were slightly modified for use on the ODROID: `https://github.com/jdourlens/FirmataC`

USING AN ODROID-XU AS A WIFI ROUTER

GET TO 802.11AC WITH STYLE

by Mauro Ribeiro, Hardkernel Developer



Believe us, this setup means business

If you'd like to use your ODROID-XU as a wireless router, you'll need the following items, all of which are available from the Hardkernel website:

ODROID-XU (+E or Lite)

USB 3.0 to Gigabit LAN Adapter

MicroUSB 3.0 to USB OTG Cable

USB 3.0 to 802.11ac Adapter

microSD or eMMC with Ubuntu Server installed

The ODROID-XU works best for this project because it offers a USB 3.0 port, so that we can use the full bandwidth of it for Wifi AC, and because the XU models also permit the use of the USB 3.0 Gigabit LAN adapter. For best throughput, we'll use our 10/100 Onboard LAN as the WAN Port.

The USB 3.0 to Wifi adapter used in thie article is a Netis WF2190, which is based on the Realtek 8812AU chip. If you have problems finding the Netis WF2190 Adapter, there is a listof equivalent adapters using the same chip here: `http://goo.gl/dNwdnY`.

Before starting, make sure that you have at least 4GB of free disk space. The following values are used in this tutorial for example purposes (your local network may differ):

The LAN address will be 10.10.10.0/24

The LAN gateway (ODROID) will be 10.10.10.254

The DHCP Range will be 10.10.10.1 to 10.10.10.253

The LAN Interface is eth1 (Gigabit Adapter)

The WAN Interface is eth0 (Connected to the internet)

The Wifi Interface is wlan2

Although unlikely, the Wifi interface value (the last item on the list) may be different for your LAN, but this is covered below. It's also very important for security that you change both the root password and the "odroid" user password, in case the board is exposed to the internet.

Install the necessary packages and dependencies

```
apt-get install git build-essential bridge-utils
```

Build the kernel

```
git clone --depth 0 https://github.com/hardkernel/linux.git -b odroidxu-3.4.y linux

cd linux

make odroidxu_ubuntu_defconfig

make -j5

make modules_install

cp arch/arm/boot/zImage /media/boot

sync && reboot
```

This builds a new -XU kernel from the latest GitHub source code. Now you have an updated copy of the kernel on your ARM Board!

Dealing with Realtek Wifi drivers.

Realtek wifi drivers are slightly tricky to configure:

Use the provided drivers

Netis ships a mini-cd with the Linux drivers inside. You'll have a linux folder with the following folder: `RTL8812AU_8821AU_linux_v4.2.0_6952.20130315`. Copy that folder to your ODROID, since these are the drivers that we need to build.

Build the wifi drivers

```
cd RTL8812AU_8821AU_linux_v4.2.0_6952.20130315/driver

tar zxvf rtl8812AU_8821AU_linux_v4.2.0_6952.20130315.tar.gz

cd rtl8812AU_8821AU_linux_v4.2.0_6952.20130315
```

* edit the Makefile and on line 583 ARCH ?= .... change to ARCH ?= arm

```
make && make install

modprobe 8821au
```

Now your Wifi drivers should be working properly.

Test the Wifi Driver

```
ifconfig wlan0 up

iwlist wlan0 scan
```

Note that if wlan0 doesn't exist, check for wlan1 or wlan2 by typing "ifconfig -a". Save this information for later, because we'll need to know the wifi card name in an upcoming step.

At this point, you'll see the list of Wifi networks in Range. You can now use your ODROID as a Wifi client to connect to other wireless networks.

Building hostapd and wpa_supplicant

Because Realtek 8812 isn't supported on Linux yet, we need to build the drivers, including hostapd, from the manufacturer's source provided on their website.

Build hostapd from the manufacturer's source

```
cd   RTL8812AU_8821AU_linux_
v4.2.0_6952.20130315/wpa_
supplicant_hostapd

tar zxvf wpa_supplicant_hos-
tapd-0.8_rtw_r6747.20130222.
tar.gz

cd   wpa_supplicant_hostapd-
0.8_rtw_r6747.20130222/hos-
tapd
```

Before building and installing hostapd, let's just make sure that we don't have Ubuntu's hostapd installed:

```
apt-get remove hostapd

make && make install
```

Building wpa_supplicant

```
cd   RTL8812AU_8821AU_linux_
v4.2.0_6952.20130315

cd wpa_supplicant_hostapd

cd   wpa_supplicant_hostapd-
0.8_rtw_r6747.20130222

cd wpa_supplicant
```

Again, we have to make sure that Ubuntu's wpa_supplicant isn't present:

```
apt-get remove wpasupplicant

make && make install
```

This installs all of our wifi drivers, along with the manufacturer's provided tools. Let's move on into the configuration part.

Configuration

The Ubuntu Network Bridge must be configured in order to create the integration between the Wireless and Wired connections.

```
cd /etc/network
```

You'll also need to edit the interfaces file, using the following screenshot as an example:

```
-- BEGIN --

# setup loopback interface

auto lo

iface lo inet loopback

# setup eth0 as DHCP for our
WAN

auto eth0

iface eth0 inet dhcp

# setup eth1 as manual and
leave it empty

auto eth1

iface eth1 inet manual

# Create our bridge with
eth1 on it and use the IP
10.10.10.254

auto br0
```

After rebooting, your LAN interface will be called br0 with eth1 being part of it.

Using hostapd

Since the wireless adapter doesn't have dual radios, you have to choose between the 5GHz or 2.4Ghz frequency. If you are on a network with only 5Ghz devices, which most are nowadays, use the 5Ghz config to allow a maximum speed of 800Mb/s. If you need extra compatibility with older computers, or your devices aren't 5Ghz, use the 2.4Ghz configuration shown below.

To begin configuring the hostapd service, start by creating the following file using root privileges:

```
/etc/hostapd.conf:

5GHz version

# Change this to match your
config

interface=wlan2

ctrl_interface=/var/run/hos-
tapd

# Your network name

ssid=ODROID-NET 5Ghz

channel=36

wpa=2

# Your network password

wpa_passphrase=testtest

bridge=br0

eap_server=0

wps_state=0

driver=rtl871xdrv
```

```
beacon_int=100

hw_mode=a

ieee80211n=1

wme_enabled=1

ht_capab=[SHORT-GI-20]
[SHORT-GI-40][HT40+]

wpa_key_mgmt=WPA-PSK

wpa_pairwise=CCMP

max_num_sta=8

wpa_group_rekey=86400

2.4GHz version

interface=wlan2

ctrl_interface=/var/run/hos-
tapd

ssid=ODROID-NET 2.4Ghz


channel=6

wpa=2

wpa_passphrase=testtest

eap_server=0

wps_state=0

driver=rtl871xdrv

beacon_int=100

hw_mode=g

ieee80211n=1

wme_enabled=1

ht_capab=[SHORT-GI-20]
[SHORT-GI-40][HT40+]

wpa_key_mgmt=WPA-PSK

wpa_pairwise=CCMP

max_num_sta=8

wpa_group_rekey=86400
```

You should customize the interface parameters "ssid" and "wpa_passphrase" for your network.

Auto-Start hostapd

Edit `/etc/rc.local` and add the following lines before the "exit 0" line:

```
/usr/local/bin/hostapd /etc/
hostapd.conf &

sleep 3

/sbin/brctl addif br0 wlan2
```

DNS and DHCP

Our network needs both a DNS and a DHCP Server in order to auto-configure our clients. In this case, we'll use dnsmasq since we already installed it as a dependency in the first step.

```
- mkdir /etc/dnsmasq
```

Create the file `/etc/dnsmasq.d/odroid.conf`:

```
resolv-file=/etc/resolv.dns-
masq

addn-hosts=/etc/dnsmasq/
hosts

dhcp-hostsfile=/etc/dnsmasq/
dhcp

expand-hosts

min-port=4096

stop-dns-rebind

rebind-localhost-ok

interface=br0

# Here on this line we'll
configure the LAN interface
(br0), initial IP Address

# last IP address, Network
netmask and the time that a
IP will be kept to a client

# that is known as lease
time

dhcp-range=tag:br0,10.10.10.
1,10.10.10.253,255.255.255.0
,1440m

# This line we configured the
IP address of our gateway
(the odroid)

dhcp-option=
tag:br0,3,10.10.10.254

dhcp-lease-max=255

dhcp-authoritative
```

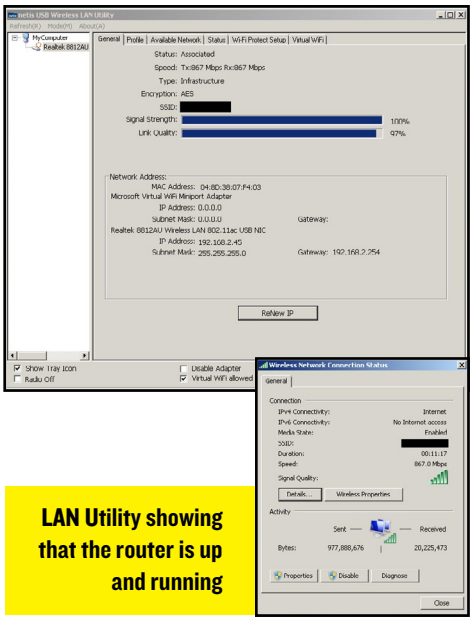
Now that our devices can connect to our new network, they need to be able to access the internet. We'll fix that right now!

Creating IP tables for internet access

Edit `/etc/rc.local`, and before "`exit 0`", add the following lines:

```
echo 1 > /proc/sys/net/
ipv4/ip_forward

iptables -t nat -A POSTROUT-
ING -s 10.10.10.0/24 -o
eth0 -j MASQUERADE
```

Now we're done, and you should have a fully functional wireless router.



LAN Utility showing that the router is up and running

To make your router even more useful, you can also attach USB disks to the USB 2.0 ports of the ODROID-XU, then create a Torrent server, a Samba file server, or any other type of shared network resource.

I've tested the ODROID-XU router with my Nexus 5 phone, and it reported a throughput of 433Mb/s. That's quite nice for Wifi, isn't it ? It could go faster, but I think that 433Mb/s is a limitation of the Nexus 5.

Next month, I'll discuss how to setup some attractive graphics to monitor the input/output data on your network.

THE ART OF MULTI-BOXING

1080p HOME MEDIA CENTER USING POCKET ROCKET AND WHISPER

by Rob Roy, Chief Editor

A pair of ODROID U3s can run all of this!

Multi-boxing is an advanced networking technique of connecting two or more computers to create a single virtual system. Originally referring to gamers who dominated online matches by controlling several players simultaneously, the methods can also be used in other applications besides gaming as a way to use parallel computing to improve the overall responsiveness of the user experience.

Compared to a single-box system, which requires a reboot to switch between platforms, the key combination Alt-Tab switches between the different desktops of the multi-boxed machines. Tasks that would normally slow down the user interface, such as downloading, video conversion, or torrents, are moved to a second computer which performs its job seamlessly in the background, but appears on the first machine as if it were a native application. As a multiple Odroid owner, it's more economical to connect several Odroids together, accessed by a single monitor and keyboard, than to purchase separate equipment for each computer.

The primary advantage of delegating tasks among several computers is that, if disk or CPU activity is high on one machine, the performance of the other systems are unaffected, since each computer has its own local resources, without needing to compete for processor cycles or hard drive access. For example, when watching videos, heavy background downloading can affect the frame rate of the video playback, causing video artifacts, frame stuttering and audio delays. Using a second system to do the work of gathering media files then frees up the main system for playing smooth video without glitches caused by spikes in disk activity.

To illustrate the concept of Odroid multi-boxing, two of the community images available on the Odroid forums, Android Pocket Rocket and Lubuntu Whisper, can work cooperatively to run two Odroids as a virtual octa-core system. Operating two images simultaneously improves the stability and efficiency of both environments, with the advantage of doubling the computing power, RAM and disk storage available for use. Alternatively, the full-featured Dream Machine image may be used instead of Whisper, and the Couch Potato SD card image may be substituted for Pocket Rocket in case an eMMC module is not available.

Gathering the Equipment

Setting up an Odroid multi-boxed virtual computer requires any two Odroids from the X, U or XU series, and a modern wi-fi router. The removable eMMC modules available from Hardkernel give the best disk performance, and for media storage, a high-capacity external USB drive or SD card permits sharing of large files between the computers without crowding the main operating system partition.

The following guide outlines the steps necessary for building a typical Home Media Center with two Odroids, using one Odroid as the media server and another Odroid as the media client. Since XBMC for Android gives the best performance in 1920x1080 resolution, I chose Android Pocket Rocket as the client, which already has the necessary SSH, VNC, and Samba protocols installed. For the media server, Lubuntu Whisper works well because of the preconfigured Desktop Sharing, Transmission and Samba applications. Both images are available for download from the Odroid forums at `http://forum.odroid.com/`.

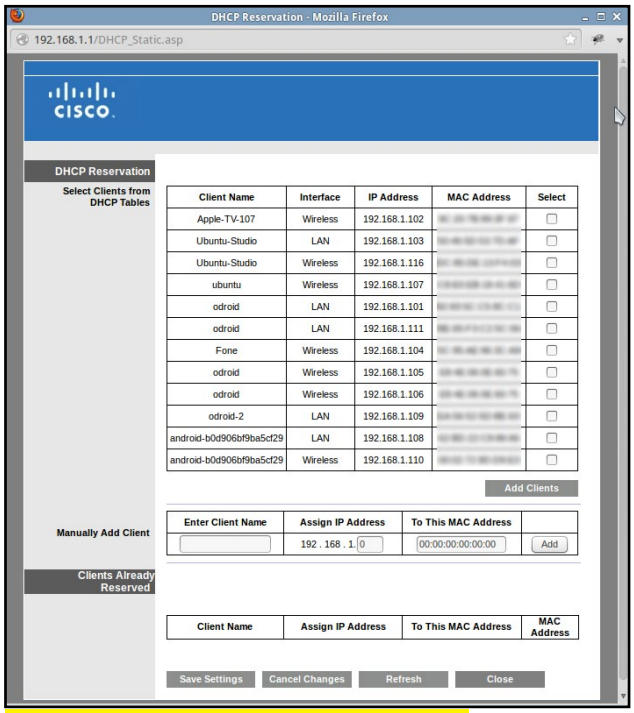
In this dual-box system, media acquisition is performed only by the Linux server, which can elegantly handle a high volume of disk and ethernet activity. When media files are ready for viewing, the Android system loads them from a common shared directory and plays them using a hardware-decoded video player like XBMC or MXPlayer.

The first step, after downloading and copying the images to an eMMC module or SD card, is to set up the network neighborhood so that the Odroids can find each other.

Configuring the Router

The server system running Whisper needs a local static IP address so that the other Odroid can locate it on the network easily. Without a static IP, the connections would need to be reconfigured with a new IP address after each reboot.

The option to assign a permanent IP address, also called DHCP reservation, is included in the wi-fi router's administrative panel. For example, using a stan-

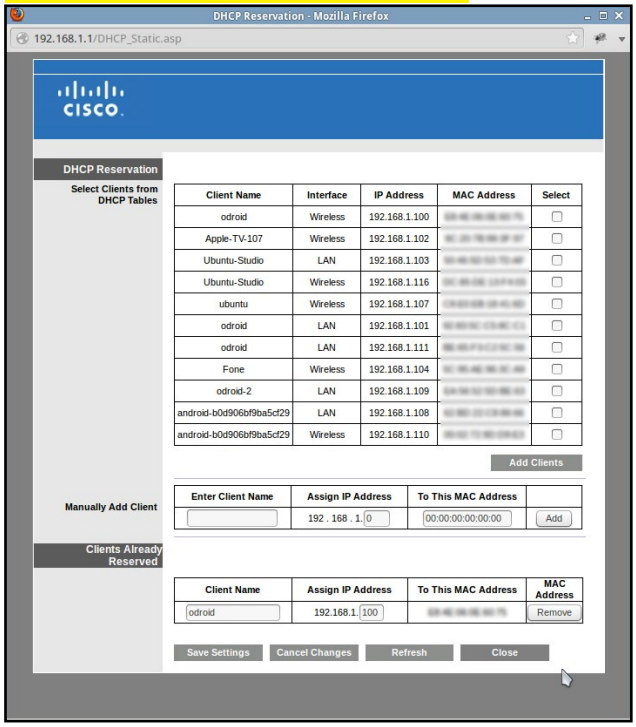


Typical router admin panel with option to assign static IP address

dard home Cisco router, the admin page is accessed at 192.168.1.1 with a default blank username and password of "admin". Some routers use 192.168.0.1, 10.0.0.1 or another similar address instead. For specific instructions, refer to the router manufacturer's website or user manual for details on accessing the admin panel and setting a static IP address.

To reserve the static IP address, boot up Whisper with its ethernet port connected to the router, then login to the router from any computer on the network. Follow the manufacturer's instructions, setting the static IP address for the Odroid running Whisper to a convenient and easy-to-remember number, such as 192.168.1.100.

Router admin panel with "odroid" client added to DHCP reservation list as 192.168.1.100



Enabling Desktop Sharing, Samba and SSH

The two Odroid have three channels of communication with each other, known as protocols. Each protocol provides a different service to allow specific types of remote access to the host server's disk, memory and peripheral resources:

VNC stands for Virtual Network Computing, a protocol that broadcasts a graphical desktop to a remote system and processes input from the remote keyboard and mouse as if they were connected locally. VNC servers and clients are available for nearly all modern operating systems, including iOS, Android, Windows, Linux and OSX. Its wide appeal permits unique combinations like accessing a Windows desktop from a Macintosh, running a Linux desktop from within Windows, or wirelessly controlling multiple Odroids from an Android phone or iPad.

SSH is a Secure SHell which creates an encrypted text-based connection between two computers using a networked terminal. With SSH, remote commands may be sent to the server using the standard command line interface to start and stop applications, reboot the system, and perform other administrative functions. In the media center example below, it will be used to launch the VNC server.

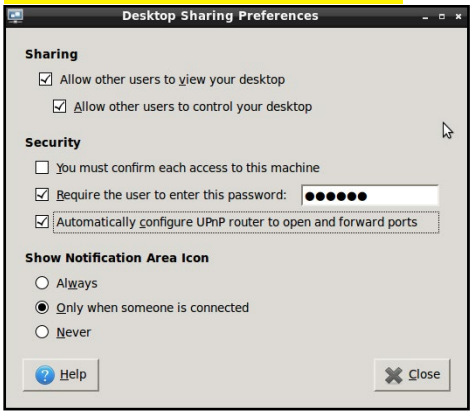
Samba is a free software implementation of the SMB/CIFS networking protocol, allowing remote file servers to be mounted as local drives. For purposes of media sharing, the Samba server enables video, audio and other files to be accessed by the Android media client via the ethernet port.

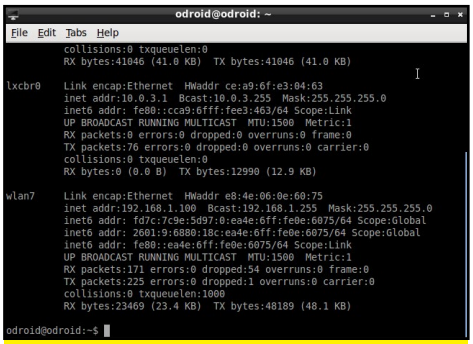
To improve system security, all three protocols require separate usernames and passwords. For convenience, Samba, SSH and VNC are initially synchronized to use "odroid" as the username and password. These passwords should be changed before moving on to the next step, but in the following screenshots, the security remains unedited and all password fields contain the default value of "odroid".

Setting Up the Server

For the initial setup, connect the Odroid running Whisper to the keyboard, mouse and monitor. Enable Desktop Sharing by selecting "Desktop Sharing" from the "Internet" menu, and matching the options to those shown below. Choose an appropriately secure password for desktop access which will later be used in setting up the Android bVNCFree client.

Desktop Sharing configuration menu





Output of ifconfig command on the Whisper server

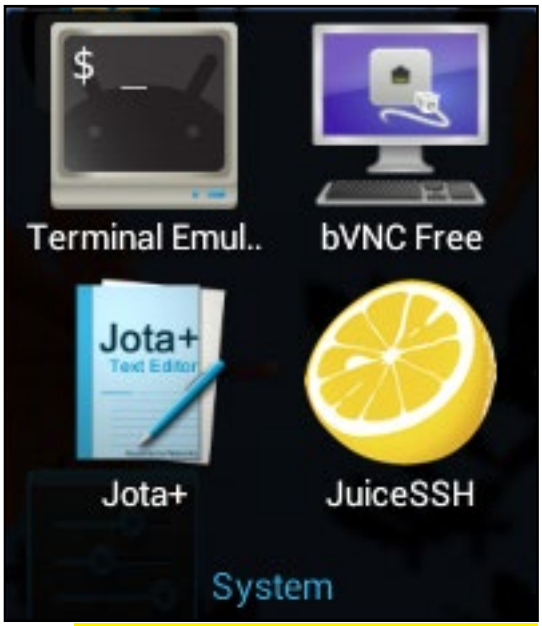
As a double-check, verify that the IP address of the Whisper system matches the one selected in the DHCP Reservation router setup by running Terminal from the Whisper desktop and typing "ifconfig". If the addresses don't match, it may be necessary to reboot Whisper in order to register it under the new IP address stored in the router.

Finally, connect any large-capacity device to use as a shared directory for media files such as an external USB drive to the Whisper server, and run the "Samba" applet from the "Preferences" menu. Match the options shown in Figures 5-8, substituting the preferred storage directory for the one shown in the screenshot. For the purposes of running the stress tests detailed below, set the shared directory to "/home/odroid/Downloads" since this is the default save directory for Transmission, Firefox and Chromium. If using an external USB drive, configure all three applications to save to a shared Samba shared storage directory located on the external drive.

Setting Up the Client

Now that the server is ready, disconnect the keyboard, mouse and monitor, and connect them to the Android Pocket Rocket client machine. From now on, the Whisper server will not need any peripherals except in the case of network or router failure. The server's desktop, command line interface, files, and CPU are all available to the Android Pocket Rocket client via the the SSH, VNC and Samba protocols.

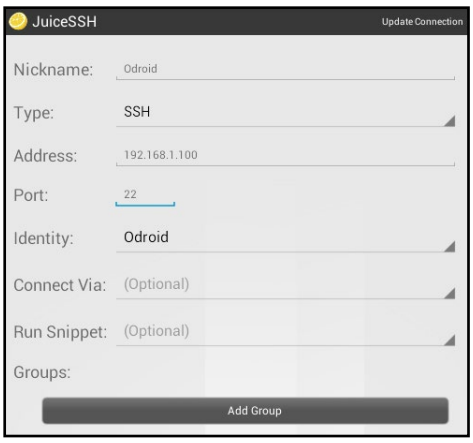
The initial connection between the two computers is via SSH, so that the VNC desktop server may be started, which then allows the X11 desktop to be shared. After the Vino server loads, JuiceSSH remains in the background while the Android application called bVNCFree gives remote control of the Whisper desktop.



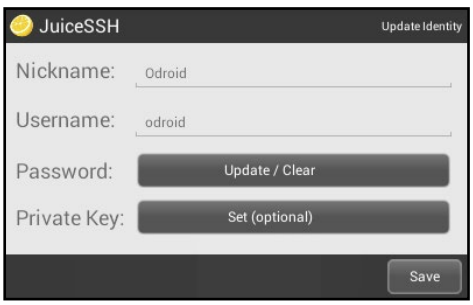
Pocket Rocket desktop contains JuiceSSH and bVNCFree, make good use of them

Connecting the Client to the Server

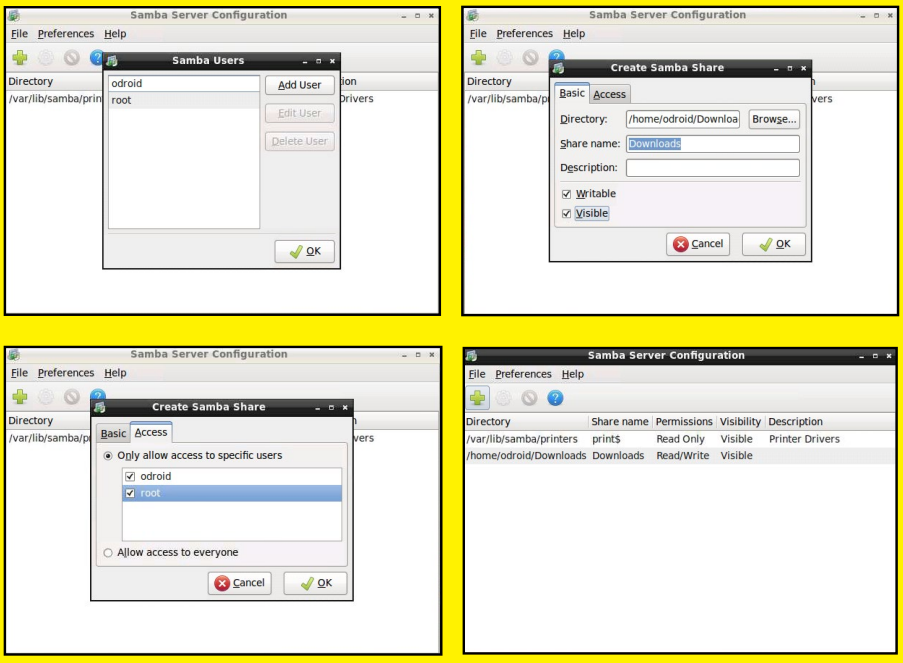
Launch JuiceSSH and select "Manage Your Connections", and match the options shown in below figures, using the static IP address registered with the router in the first step. Advanced users may skip the SSH step by adding Vino as a Linux startup task.

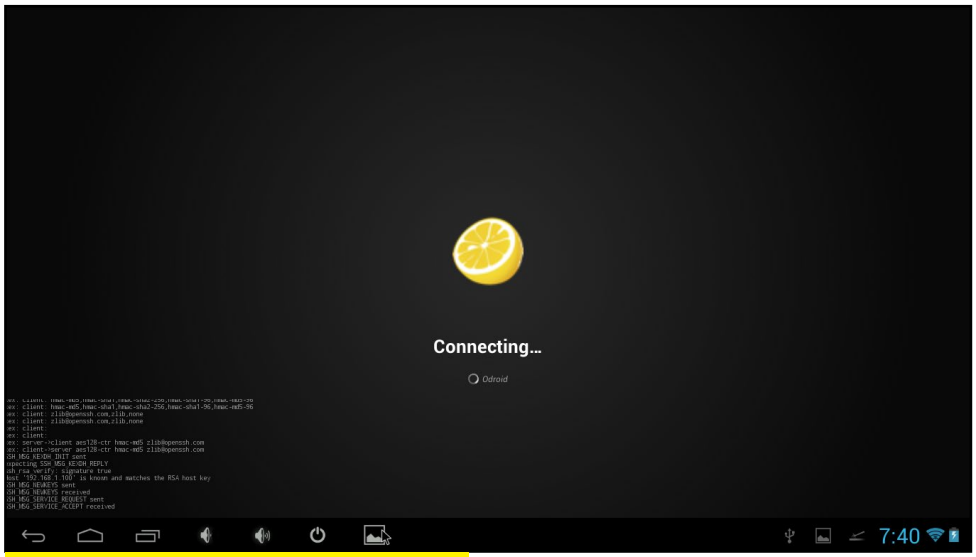


JuiceSSH configuration options



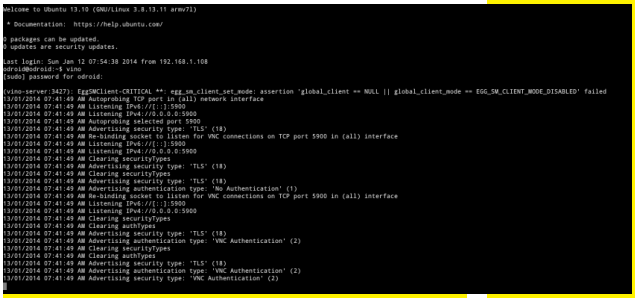
Configuring the Samba options in Whisper





Juice SSH connecting to the Whisper server

Once the server is connected, JuiceSSH drops into the remote server's scripting shell, known as the command line interface or command prompt. For convenience, Whisper includes a local Vino script which automatically shares the current desktop using a single command. Launch the VNC server by typing "vino" at the SSH command prompt, and entering the root password of "odroid".



JuiceSSH after running the "vino" command

Keyboarding Like a Pro

Before moving on to using the graphical X11 desktop, it's important to become comfortable with the Android interface itself. To optimize the user experience, Android provides several keyboard shortcuts that permit quick window management and data sharing functions while also alleviating the need for using the mouse.

Android has many built-in shortcut key combinations that closely resemble those used in Windows and OSX.

Esc: Close a popup window

Control-W: Close the current window

Control-T: Open a new browser tab

Control-C: Copy

Control-X: Cut

Control-V: Paste

Control-A: Select all

Tab: Navigate to next input field

Alt-Tab: Switch to another application

Alt-Tab -> Alt-Esc: Return to the desktop (same as Home button)

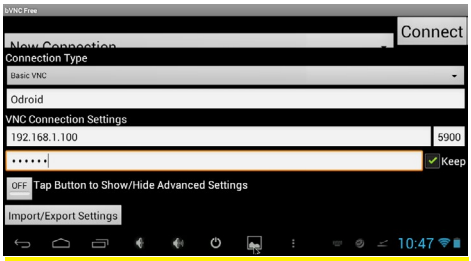
Mouse Right-click: Go to previous window or close a popup window if there's one open

As an example of keyboard navigation, you can press Alt-Tab, then Alt-Escape to get back to the Android desktop instead of clicking on the Home button with the mouse from within any Android application.

The Alt-Tab menu in Android Pocket Rocket

Connecting with VNC

Launch the bVNCFree application, and select "New Connection". Match the configuration options to those shown in Figure 8, again using the static IP address registered with the router and used when setting up JuiceSSH.



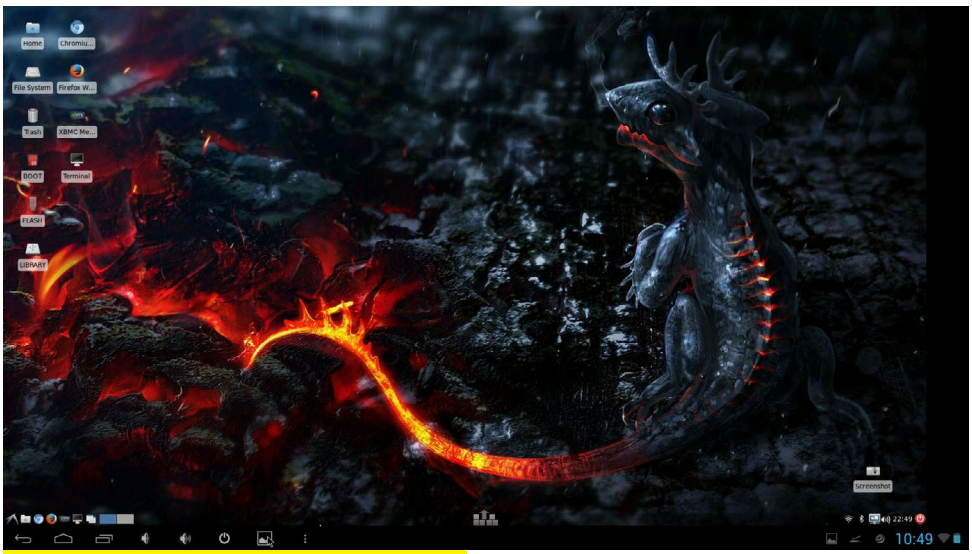
Configuration options for bVNCFree

lished, the Whisper desktop opens up, allowing the Android client to take full control of its storage, CPU and resources. By also enabling the "Fit To Screen" scaling option, the virtual system gives the appearance that the server desktop is running natively on the local Odroid client while simultaneously running Android applications.

Running the Media Server

Now that the two systems are connected, they are ready to perform some tough tasks to show off how well the networked machines handle a high volume of disk I/O. For comparison, Pocket Rocket will do all of the work on its own as an example of single-boxing. By stress testing the client system, the areas of improvement become more apparent when many large files are processed at the same time.

For the single-box test, download several files, start a few torrents, and watch an XBMC video using only Pocket Rocket. Although the Android system performs well under medium to heavy load, there will be brief moments during video playback where the screen will stutter, or the audio will be out-of-sync due to spikes in disk access. Because the eMMC or external drive has many applications competing



The Whisper desktop, as viewed through the Android bVNCFree application

for storage, the playback experience gets slightly degraded at random intervals due to locked resources.

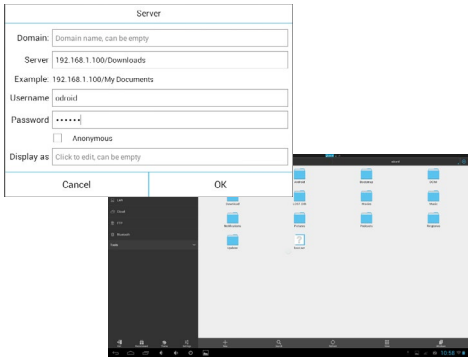
To improve stability and video playback, the tasks of downloading and handling high volumes of disk access are delegated to the Whisper server in the dual-box setup. This separation frees up the processor cycles and disk usage on the Android client so that the video player isn't waiting for another application to finish before loading the next piece of the video from the hard drive.

In order to perform the dual-box stress test, shut down all of the downloads and torrents on the Android system so that only the video player is running. Press Alt-Tab to switch to the bVNCFree application, then use the Whisper desktop to download several large files and torrent a few more, using the web browser and Transmission.

After one of the media files has finished downloading, press Alt-Tab to switch to the Android video player, such as XBMC, then play the file from the newly added Samba shared directory.

The simplest way to load a media file, is to log into the Samba share using either ES File Explorer's "LAN" tab and single-click a media file from the shared directory, or by starting XBMC and logging into the Samba share using the XBMC interface.

Notice that moving the downloads to the Whisper server allows the videos to play smoothly on Android, without stuttering or momentary freezing. The playback on the Android client remains consistent despite heavy stress,on the server, which grealy improves the experience of using Pocket Rocket by itself.



ES File Explorer's LAN tab

Peeking Under the Hood

Since the Android operating system and video player have no competition for the local hard drive, overall performance improves on the Android client since it has immediate access to the hard disk, now that the media files streams have been moved over to the ethernet port. The result of the coordination between the two systems is that the Android client remains responsive at all times, while the Whisper server accomodates the increased disk activity in the background without locking the Android system's resources. The dual-box version of the Media Center performs better than the single-box version because of its compartmentalized use of each system's peripherals.

Files may also be copied and shared from one system to another using ES File Explorer, so that documents may be edited locally on the Android client system while maintaining a read-only master copy on the Whisper server. Additionally, bVNCFree allows text data to be transferred between previously unconnected desktop environments (ex. between Windows and Linux) using the Copy and Paste function of the Android desktop, instead of resorting to an intermediate file or email.

Building a Cluster of Odroids

Follow the steps again to connect more computers to the cluster, with a separate SSH and VNC session controlling each additional system. For my personal multi-boxing system, I connected several Odroid computers which offer a dedicated web server, application development packages, and a security testing suite, all operating in their own resource spaces, coordinated using the Android Pocket Rocket desktop.

Advanced Multi-boxing

Seasoned Linux users can add automation to the Media Center by scheduling cron jobs, adding RSS feeds, and kickstarting downloads using the rTorrent command line interface. Further customizations include enabling the web interface in XBMC and Transmission for true remote management, and installing a local Apache website for managing, sorting and displaying the media library contents in an easy-to-read format. Handbrake has also recently been ported to the Odroid, which can be included in the download process for converting media files to the preferred resolution and codec before making them available to the Android client.

MEET AN ODROIDIAN

MAURO RIBEIRO: THE SOFTWARE GENIUS BEHIND ODROID'S LINUX KERNELS

edited by Rob Roy

What is your official title and role at Hardkernel?

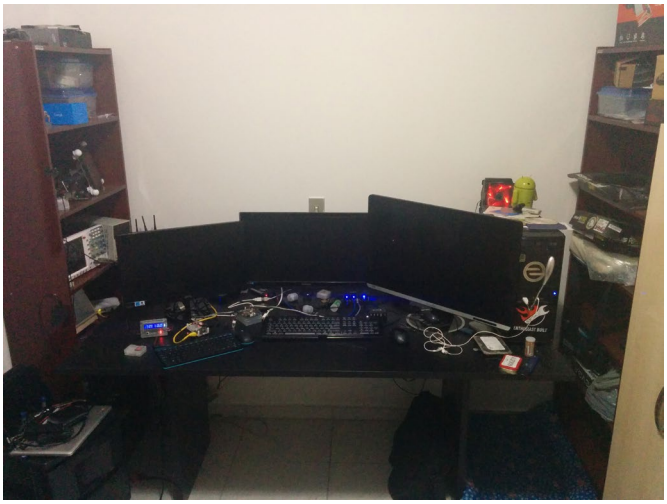
Senior Software Engineer, I mainly do Kernel work, Linux Support and deal with our forums. Trying to get onto the Hardware world too.. Its a lot of fun :)

As one of the moderators of the ODROID forums, what do you like most about the ODROID community?

I think that what I like most the is amount of information that we managed to share and how many users that we had with no Linux knowledge, now they are quite advanced.

How did you get started with computers?

Since I was 2 years old (from what I can remember), I've shown a tendency in dealing with electric and electronic stuff. Mostly because of my uncle's influence, who taught me basic electronics since I was kid. I was 5 or 6 years old when my uncle showed me an Apple 2+. From a friend of his. This was on 1989 or 1990.



A year later, my uncle gave me an awesome 386 computer. I still recall the exact specs" 386-DX40, 4MB of RAM, VGA Card, 14" Color Monitor (CRT), Dual Floppy 5 1/2 and 3 1/4. And an outstanding 260MB Seagate HDD. I kept this machine. The CRT and HDD are dead, but everything else works.

I really got into Linux in 1998. A friend of mine brought me a CD-ROM from USA containing "Slackware 97". That is where everything started.

What are some of your favorite projects that you've seen ODROIDs being used for?

I love projects that deal with light control and ambiance light control. I don't think I have seen any of those so far using ODROID's. I've seen some pretty neat robots powered by ODROID's, and I'm on my own personal project to do some light control with ODROID's :)

Do you have any personal projects that you're working on using ODROIDs?

Yes, ambient light control. It's using a U3+Shield board to control the light on my room using 5 10W leds. Right now I have it about 70% complete. Minor bits are missing, such as cabling, housing and remote control Software.

Mauro dressed in a typical korean attire, a cool memento of his last korean trip.

What other interest and hobbies do you enjoy?

Race Cars are probably what I love most apart from the computer side. I've been part of the scene dealing with cars ECU's. Combustion engines are fun.

About my dogs: They are two Lhasa-Apso. Meg is the older, she's 6 years old now.

Junior is the last born of her mother, he's 4 years old now. Laika and Res were my very first dogs. They are the couple that gave birth to all of my current dogs. Unfortunately, Laika passed away early last year due to complications of a surgery to remove stones on her bladder. Res sadly passed away a couple months later. Laika died at age of 7 and Res at age of 9.