Debian, Tizen, Lakka and Android Marshmallow for XU4 • XU4 Case

ODROID Magazine

Our 3rd year!

Year Three
Issue #25
Jan 2016

For Serious Gamers

Hardkernel presents its new Universal Motion Joypad

LINUX GAMING

- Kernel Compilation
- XU4 CPU Control
- ODROID Universal Installer

What we stand for.

We strive to symbolize the edge of technology,
future, youth, humanity, and engineering.

Our philosophy is based on Developers.
And our efforts to keep close relationships with
developers around the world.

For that, you can always count on having the quality
and sophistication that is the hallmark of our products.

Simple, modern and distinctive.
So you can have the best to accomplish
everything you can dream of.

HARDKERNEL

Pollin ELECTRONIC

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

Address: Max-Pollin-Straße 1
85104 Pförring Germany

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

Our ODROID products can be found at
http://bit.ly/1tXPXwe

EDITORIAL

ODROID Magazine is celebrating the beginning of its third year! All of our contributors constantly amaze us with their innovations. Each month, our regular columnists like Tobias, Venkat and Nanik write fantastic in-depth tutorials and reviews that help our readers find new ways to work with and enjoy their ODROIDs. We look forward to continuing to produce the best magazine that we can. I hope that the worldwide ODROID community continues to send us great and fascinating articles, since that's what really makes our publication thrive.

This month, we present the new Universal Motion Joypad to make driving games more fun. Tobias offers a set of strategy games that are sure to keep you entertained, and we highlight a few innovative operating systems such as Tizen, Lakka, Debian Jessie Server, and Android Marshmallow. We also showcase a modern aluminum case for the ODROID-XU4, teach you how to throttle your CPU to reduce fan noise, detail the steps for compiling a kernel, and introduce a convenient way to install a new single or dual-boot operating system. Don't forget to check out our new website at http://magazine.odroid.com, now featuring a searchable master article list.

ODROID Magazine, published monthly at http://magazine.odroid.com, is your source for all things ODROIDian.
Hard Kernel, Ltd. • 704 Anyang K-Center, Gwanyang, Dongan, Anyang, Gyeonggi, South Korea, 431-815
Hardkernel manufactures the ODROID family of quad-core development boards and the world's first ARM big.LITTLE single board computer.
For information on submitting articles, contact odroidmagazine@gmail.com, or visit http://bit.ly/1yplmXs.
You can join the growing ODROID community with members from over 135 countries at http://forum.odroid.com.
Explore the new technologies offered by Hardkernel at http://www.hardkernel.com.

HARDKERNEL

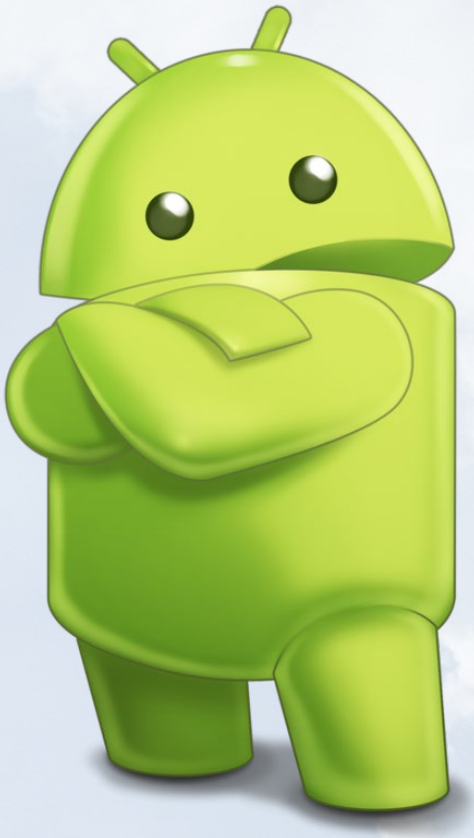
ODROID Magazine

Rob Roy, Chief Editor

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

Manuel Adamuz, Spanish Editor

I am 31 years old and live in Seville, Spain, and was born in Granada. I am married to a wonderful woman and have a child. A few years ago I worked as a computer technician and programmer, but my current job is related to quality management and information technology: ISO 9001, ISO 27001, and ISO 20000. I am passionate about computer science, especially microcomputers such as the ODROID and Raspberry Pi. I love experimenting with these computers. My wife says I'm crazy because I just think of ODROIDs! My other great hobby is mountain biking, and I occasionally participate in semi-professional competitions.



Bruno Doiche, Senior Art Editor

Bruno spent a couple of months sniffing networks and working like mad to get all of our publications on target. As usual, it was crazy. But crazy fun!

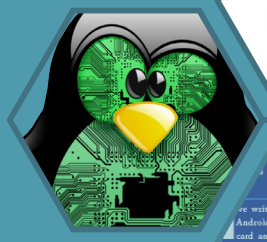
Nicole Scott, Art Editor

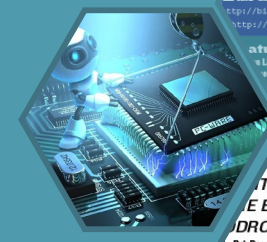
Nicole is a Digital Strategist and Transmedia Producer specializing in online optimization and inbound marketing strategies, social media management, and media production for print, web, video, and film. Managing multiple accounts with agencies and filmmakers, from web design and programming, Analytics and Adwords, to video editing and DVD authoring, Nicole helps clients with the all aspects of online visibility. Nicole owns anODROID-U2, and a number of ODROID-U3's and looks forward to using the latest technologies for both personal and business endeavors. Nicole's web site can be found at http://www.nicolecscott.com.

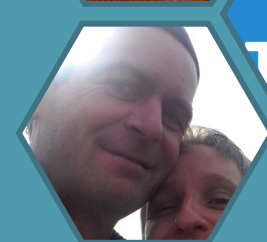
James LeFevour, Art Editor

I'm a Digital Media Specialist who is also enjoying freelance work in social network marketing and website administration. The more I learn about ODROID capabilities, the more excited I am to try new things I'm learning about. Being a transplant to San Diego from the Midwest, I am still quite enamored with many aspects that I think most West Coast people take for granted. I live with my lovely wife and our adorable pet rabbit; the latter keeps my books and computer equipment in constant peril, the former consoles me when said peril manifests.

INDEX



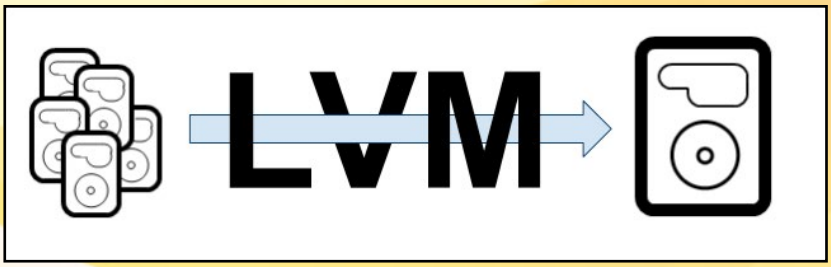




THINLY-PROVISIONED LOGICAL VOLUMES

THE ABILITY TO DYNAMICALLY ALLOCATE CAPACITY

by David Gabriel



When you start having multiple applications running on a single system, administering the storage allocated to each of them starts getting a little complicated. You typically size it to be enough to run for some time and include a generous buffer space to avoid running out of storage when it's most needed. However, some file systems' needs tend to increase quickly over time, like databases, while others grow slowly, like a web server. This results in having large amounts of storage being allocated when it is not always being fully used but still being tied to a particular volume. This kind of model is known as thick provisioning and may be inefficient sometimes.

For cases like that, we can take advantage of another great feature that LVM offers called thin provisioning. It provides the ability to dynamically allocate capacity to the volumes as required. Usually, the blocks are written as soon as you create the logical volume. In a thin provisioned logical volume, the blocks are allocated as they are written. This way, we can have an LV with a virtual size that is much larger than the physical available storage. It can then be increased as needed.

Thin provisioning can be compared to a bank. The bank holds money from all of its depositors, but all that does not exist all at one time. If every single person decides to withdraw their money at the same time, the bank will not have enough available to provide all of it to the clients. It would have to get the cash from its investments and loans, or by selling shares. Another example is an insurance company. There are a certain amount of people insured, but not all of them make claims every year. So there is a buffer to pay these claims, but the company doesn't have to have all of the money to cover the insurees at the same time. This also applies to other resources like memory and CPU on demand, especially when using virtual machines, but we will concentrate on the storage use case.

In order to create a thin provisioned volume, first a thin pool LV must be created, which comes from the combination of two LVs. One large data LV that will hold the blocks for the LV and a metadata LV that will hold metadata. The metadata LV will keep track of which block belongs to which LV inside the pool along with a few other pieces of information.

To create the thin pool, type the following commands:

```
$ lvcreate -n thinpool -L 10G rootvg
$ lvcreate -n metathinlv -L 500M rootvg
$ lvconvert --thinpool rootvg/thinpool \
  --poolmetadata rootvg/thinmetalv
```

The first two commands create the data and metadata LVs, and the last is used to merge them into a thin pool LV. Then, you can create the LVs you want from inside the pool:

```
$ lvcreate -n mysqllv -V 1T \
  --thinpool rootvg/thinpool
```

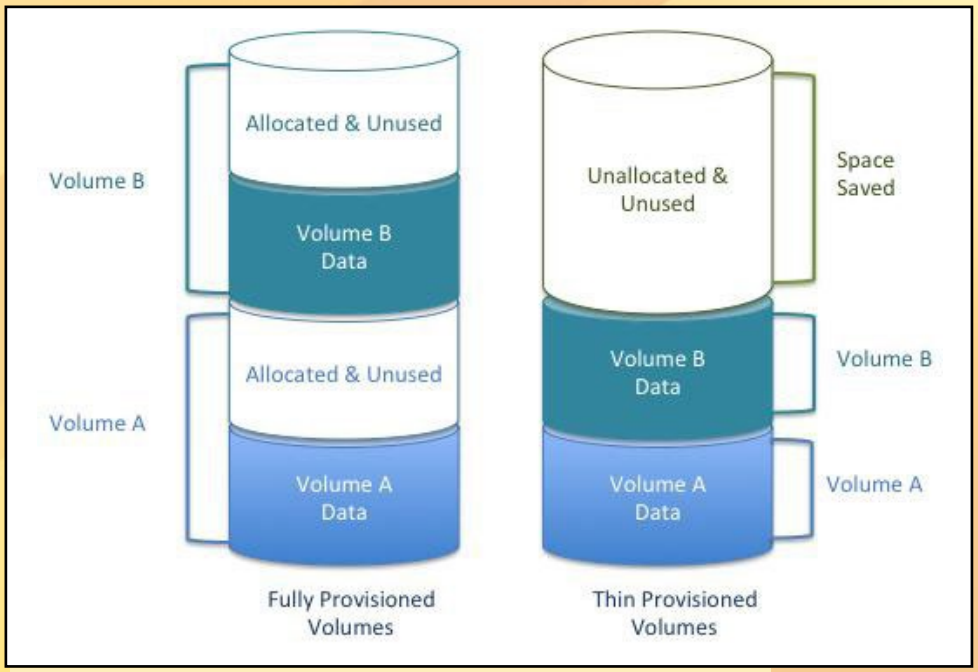
You can check your newly created LV with the lvs command. This will create a new LV from the thin pool with 1TB. Notice that we clearly don't have 1 TB of available space, since the thin pool has only 10GB, but the system will let us create that virtual space because the blocks will be allocated as needed. If you don't want to worry about the metadata part you can create the thin volume in a single step:

```
$ lvcreate --thinpool thinpool -L 10G rootvg
```

Or even create the pool and an LV inside it:

```
$ lvcreate -L 10G -V 1T -n mysqllv \
  --thinpool rootvgvg/thinpool
```

You may have already figured this out, but you should have the space usage on thin provisioned file systems be very con-

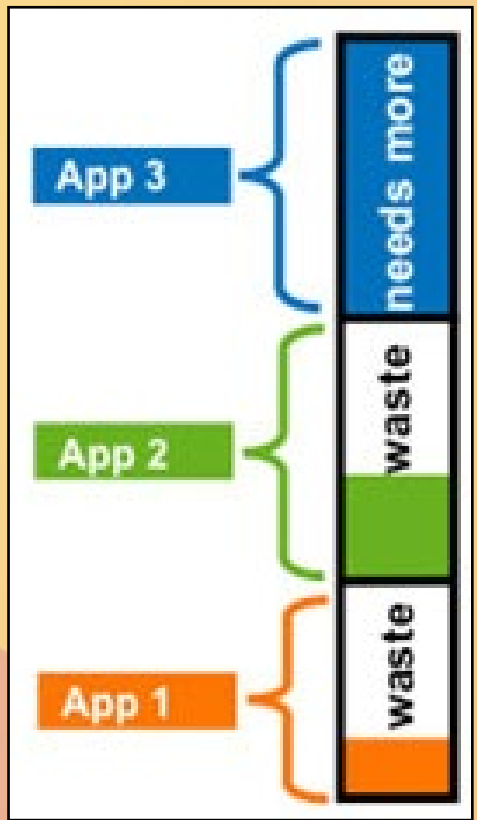


A quick comparative of Fully Provisioning versus Thin Provisioning

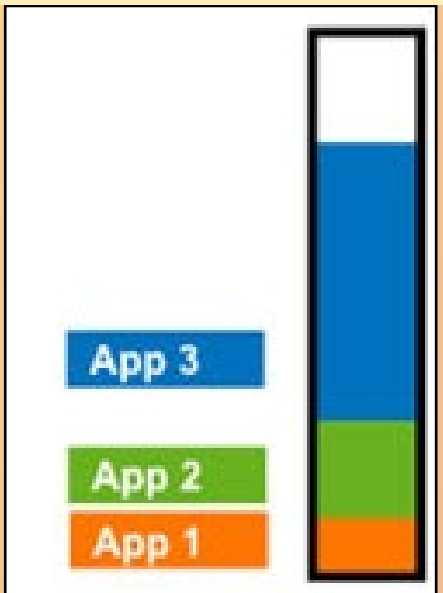
trolled. If possible, set alerts that go off when the allocation increases too much in order to avoid over-allocation, which may result in data corruption. Once the usage starts to grow, you must extend the thin pool LV just as you would on a standard LV. The system will try to write on a block that does not exist, so you should be very careful when using thin provisioning.

The storage allocation from thin pools may cause the LVs to become fragmented. Standard LVs generally avoid this problem by allocating a full single block on disk. Unless you have a really big thin pool, this should not impact the disk performance over time.

Thin provisioning can be great if used wisely. It avoids having space allocated without being fully utilized, so that blocks will go to where they are needed. However, if the administrators don't keep an eye on it, it might turn into a really bad headache.



You can do more with less and be flexible



DEBIAN JESSIE FOR ODROID-XU4

A MINIMAL SERVER IMAGE

by Tobias Schaaf

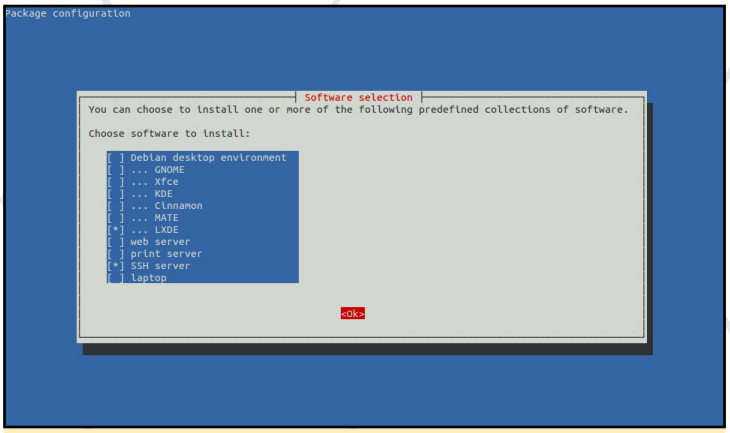
I recently released a minimal Debian Jessie image that can be used as a configurable server or desktop. The image is a basic headless server with only the root user. However, it has all my repositories already added, which allows for easy installation and updates of packages such as a different kernel, Kodi, Chromium Browser, or any number of other packages. The image file is available `http://bit.ly/1mf2CcD`, and is compatible with both microSD card and eMMC module. The total footprint is 99MB compressed and 472MB uncompressed.

On first boot, the image will resize the root file system partition and configure SSH, then automatically reboot after the initial setup. At that point, the image is ready to use. The kernel and headers are already installed if you need to build your own drivers. A few basic tools such as ntp, htop, mc, vim, and bash-completion are also available for convenience.

Adding a desktop

Since all of my repositories are available with pre-built packages for Debian Jessie, it's easy to convert this image to a desktop image using the following steps. First, update the package lists:

```
# apt-get update
```



Tasksel allows easy installation of many software packages

Next, run tasksel in order to choose your preferred desktop environment, as shown in Figure 1:

```
# tasksel
```

Please note that not all Debian desktop environments work perfectly on the ODROID. The best choices are LXDE or MATE, but XFCE or KDE should work as well. Tasksel will take quite some time to download and install all the packages needed for a desktop image. It also requires at least 1GB of extra disk space, although 2GB is recommended.

Once the desktop environment has finished installing, we need to install X11 framebuffer drivers:

```
# apt-get install xf86-video-armsoc-odroid
```

Mali GPU drivers for 3D acceleration may also be installed:

```
# apt-get install malit628-odroid
```

You will need an appropriate xorg.conf for the framebuffer drivers:

```
# cd /etc/X11
```

```
# wget \
  http://oph.mdrjr.net/meveric/other/xorg.conf
```

If you are a more experienced user, you can always install the packages you want manually and therefore keep the image as slim as you want instead of using tasksel. It's also recommended to create a new user account for the desktop environment instead of logging in as root:

```
# adduser odroid
```

Once all of the installation steps are completed, reboot the ODROID. After a few minutes, you should see a graphical login for your Debian Jessie image.

Once you have your desktop running, you can easily install all the packages available in my repository such as Kodi or XBMC.

To install Kodi type the following into a Terminal window:

```
# apt-get install kodi-odroid
```

To install XBMC instead, type the following into a Terminal window:

```
# apt-get install xbmc-odroid
```

Next, install the firmware required for hardware decoding in XBMC and Kodi:

```
# apt-get install firmware-samsung
```

The TVHeadend package allows you to watch live TV using Kodi:

```
# apt-get install tvheadend
```

An optimized version of the Chromium browser may also be installed from my repository:

```
# apt-get install chromium-browser-odroid
```

Arduino IDE is necessary for development with the ODUINO board, ODROID SHOW, and other Arduino-based electronic devices:

```
# apt-get install arduino
```

I also recommend installing ffmpeg from the Debian repository, which is a very good tool for watching and converting videos, and is maintained by experienced Debian developers.

Notes

Most of the packages available in my repository are X11-based. For example, the malit628-odroid package are X11 GPU drivers, and Kodi is only available for X11, so you need a desktop environment to use either of those package. Also, I haven't installed all of the drivers and firmwares available. If you want to use the Wifi Module 4, you will have to install the firmware-ralink package available from the standard Debian repository.

Changing language

If you set up a desktop environment, you should already have a keyboard configuration installed. Debian will ask upon the first installation which keyboard layout to use for your X11 desktop. For your console, you also need to install console-setup:

```
# apt-get install console-setup keyboard-configuration
# dpkg-reconfigure keyboard-configuration
```

You will probably want to set the timezone as well:

```
# dpkg-reconfigure tzdata
```

CEC support

For CEC support, you will need to install the libcec package:

```
# apt-get install libcec
```

If needed, you can also install the cec tools type:

```
# apt-get install cec-utils
```

After that, you need to add a new udev rule so that you can access the CEC device:

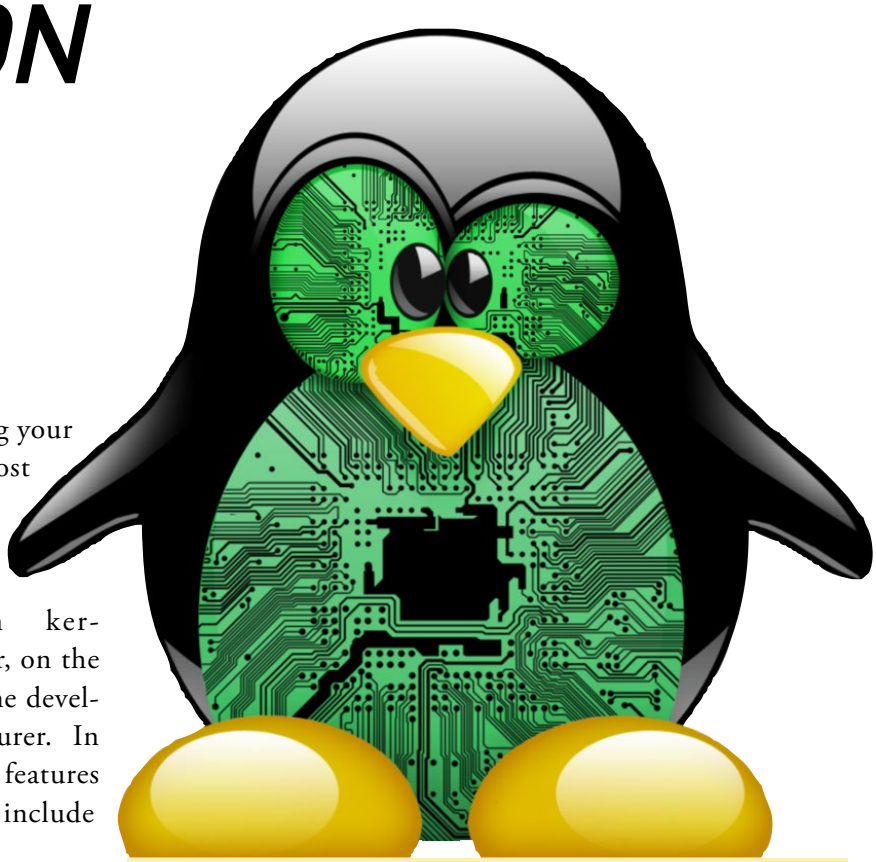
```
# echo 'KERNEL=="CEC",SUBSYSTEM=="misc",MODE="0666"'\
  > \ /etc/udev/rules.d/20-hkl_cec.rules
```

If you find any bugs please let me know, and if you have another model besides the ODROID-XU3 or XU4, I can easily convert this image for use with the ODROID-X, X2, U2, U3, or C1. If you have comments, questions or suggestions, please visit the original thread at `http://bit.ly/1k5qldS`.

LINUX KERNEL COMPILATION

HOW TO CUSTOMIZE YOUR OPERATING SYSTEM

by Uli Middelberg



Who wouldn't want to completely control your own kernel? We certainly do!

This tutorial covers some aspects of compiling your own Linux kernel for your ARM device. Most Linux distributions for the x86 PC platform maintain a Linux kernel which supports a broad range of hardware devices, so it has become very unlikely to need to compile your own kernel from source when using x86 devices. However, on the ARM platform, the Linux kernel is provided by the development board or system on chip (SoC) manufacturer. In many cases, these kernels include a minimal set of features and device drivers only and need to be modified to include more functionality.

Additionally, you may want to include a specific feature set which is provided as a patch to the kernel sources only, such as enhanced security (bit.ly/1wcJIa3) or real-time capabilities (bit.ly/1OQlDcv). Some use cases impose special requirements. For example, you may prefer to switch off loadable kernel module support in high security environments and build a monolithic kernel instead. Or, you may have to accommodate restricted resources by building a very tiny kernel image.

Recent ARM devices have become quite powerful, so I prefer to compile the kernel on the target device directly instead of cross compiling, to reduce the level of complexity.

Kernel components

The Linux kernel consists of the following components:

- **Kernel image <boot-partition>/zImage or <boot-partition>/uImage, depending on your u-boot capabilities and configuration**
- **Device tree binary, a low level device description, specific to your device <boot-partition>/<board>.dtb**
- **Kernel modules /lib/modules/<kernel-version>/***
- **Device firmware /lib/firmware/***

These components are built from the kernel sources with the help of the make utility. Usually, the kernel image and the device tree binary are loaded from a small vfat boot partition, which is mounted as /boot or /media/boot. The rest reside in the root file system.

Prepare the build environment

Besides the make utility, several other utilities are needed to compile the Linux kernel. For example, with Ubuntu, you will need to install the following packages:

```
$ sudo apt-get -y install \
bc curl gcc git libncurses5-dev \
lzop make u-boot-tools
```

Set the default compiler

At the time of this article's publication, most Linux distributions have updated their default compiler to gcc version 5. If your distribution still uses gcc version 4.8 as a default, you should consider switching to gcc version 5. Ubuntu 14.04, for example, uses 4.8. To update to gcc version 5, type the following commands in a Terminal window:

```
$ sudo apt-get -y install \
  python-software-properties;
$ sudo add-apt-repository -y \
  ppa:ubuntu-toolchain-r/test;
$ sudo apt-get update
```

```
$ sudo apt-get -y install gcc-5 g++-5
$ sudo update-alternatives --install \
  /usr/bin/gcc gcc /usr/bin/gcc-5 50
$ sudo update-alternatives --install \
  /usr/bin/g++ g++ /usr/bin/g++-5 50
```

The command update-alternatives helps you to define the default command to be executed if different versions of the same command are installed at the same time. You can check the gcc version by typing the following:

```
$ gcc --version
```

U-boot

Compiling a custom kernel always comes with the risk that your new kernel won't boot for a number of reasons. I recommend defining a u-boot macro that will load and boot a test kernel before overwriting the existing default kernel. You can find more information about u-boot at bit.ly/1LMcDHM.

Download the sources

The website www.kernel.org offers the mainline Linux kernel sources for download:

```
$ curl -sSL https://cdn.kernel.org/pub/linux/\
  kernel/v4.x/linux-4.3.tar.xz | \
  unxz | tar -xvf -
```

You can obtain the most recent releases from that website, but only a few ARM devices are able to boot an unmodified mainline kernel. Even if they successfully start, it is very likely that some devices lack full support due to missing drivers, such as graphics acceleration.

To get a kernel with extensive support for your ARM board, you need to fetch the kernel source provided by the board vendor. Often, this kernel source contains additional patches for a kernel version with long term support. Many vendors, such as Hardkernel, use Github to provide and manage their specific kernel source files, which makes it very easy to add back own contributions. Usually, the vendors organize the kernel sources for each particular board into designated Github branches.

The git clone command creates a local copy of the repository:

```
$ git clone --depth 1 --single-branch \
  -b <branch> \
  <URL to the repository>
```

This local copy contains only the branch specified by <branch> with no information about prior commits. The "--depth 1" argument reduces the download size by limiting the commit tree to the most recent. For example, if you type the following, the latest revision of the kernel source code for the ODROID-C1 will be saved to a directory called "linux":

```
$ git clone --depth 1 --single-branch \
-b odroidc-3.10.y \
https://github.com/hardkernel/linux
```

Build the custom kernel

Now, you're ready to start building your own kernel. After downloading and extracting the kernel source, you start by creating a configuration file named .config. This text file contains the relevant parameters for your kernel, with one line per kernel option.

```
cd linux
make <default_config>
less .config
```

You can find the default configuration available for your ARM device in the directory ./arch/arm/configs/. These are configuration files for the boards mentioned above:

Device	default configuration
ODROID C1(+)	odroidc_defconfig
ODROID U3	odroidu_defconfig
ODROID XU3	odroidxu3_defconfig
ODROID XU4	odroidxu4_defconfig

Once the kernel configuration .config has been created, you can modify it either with a text editor, or by using the following command to change the kernel configuration interactively:

```
$ make menuconfig
```

If enabled as a configuration option, you can read the current kernel configuration of a running kernel:

```
$ cat /proc/config.gz | gunzip | less
```

When you are done with the kernel configuration, create a new default configuration:

```
$ make savedefconfig
```

This command creates a file called defconfig from the .config file, which contains only the changes with respect to the global kernel configuration defaults and reduces the file size to about 15% to 20% of the original .config file.

The next step is to build the the device tree binary, kernel modules, and kernel image, which is either a uImage or zImage file. This is the most time-consuming step, and even with parallel execution using the -j4 option, it takes about one hour to compile the C1 kernel on the C1 itself, and 20 minutes on the XU4. To start the compilation, type the following command:

```
$ make -j4 zImage uImage dtbs modules
```

Install the custom kernel

As mentioned above, you may want to test your new kernel before replacing your existing one. This step requires some knowledge about u-boot and the particular u-boot configuration for your board. You need to define a u-boot macro which boots your custom kernel instead of the system default. When you're done with testing, you can install the new kernel as the system default with these commands:

```
$ sudo cp ./auch/arm/boot/[u|z]Image \
./auch/arm/boot/dts/*.dtb <boot-partition>
$ sudo make modules_install
$ sudo make firmware_install
```

The kernel image and the device tree binary are installed in the boot partition, whereas the kernel modules and the device firmware are copied to the root file system. If you are running different Linux installations on different partitions of your eMMC or SD storage device with the same kernel image, you also need to install the kernel modules and device firmware on each of the partitions with the following commands, repeating them for each partition:

```
$ sudo make modules_install INSTALL_MOD_PATH=<path>
$ sudo make firmware_install INSTALL_FW_PATH=<path>
```

You can get a list of all make targets and parameters by typing the following:

```
$ make help
```

Examples

ODROID-C1 and ODROID-C1+

```
bit.ly/1SgNPut

$ git clone --depth 1 --single-branch \
-b odroidc-3.10.y \
https://github.com/hardkernel/linux
$ cd linux
$ make odroidc_defconfig
$ make -j 4 uImage dtbs modules
$ sudo cp arch/arm/boot/uImage \
arch/arm/boot/dts/*.dtb /media/boot
$ sudo make modules_install
$ sudo make firmware_install
```

ODROID-C1 Mainline (Experimental)

```
bit.ly/1ZuG5XK

$ curl -sSL \ https://cdn.kernel.org/pub/linux/\
kernel/v4.x/testing/linux-4.4-rc2.tar.xz | unxz | \
tar -xvf -
$ cd linux
$ make multi_v7_defconfig
$ make -j 4 LOADADDR=0x00208000 \
uImage dtbs modules
$ sudo cp arch/arm/boot/uImage \
arch/arm/boot/dts/*.dtb /media/boot
$ sudo make modules_install
$ sudo make firmware_install
```

ODROID-U3

```
bit.ly/1kl4Fue

$ git clone --depth 1 --single-branch \
-b odroid-3.8.y \
https://github.com/hardkernel/linux
$ cd linux
$ make odroidu_defconfig
$ make -j 4 zImage dtbs modules
$ sudo cp arch/arm/boot/zImage \
arch/arm/boot/dts/*.dtb /media/boot
$ sudo make modules_install
$ sudo make firmware_install
```

ODROID-XU3

```
bit.ly/1YIToBI

$ git clone --depth 1 --single-branch \
-b odroidxu3-3.10.y \
https://github.com/hardkernel/linux
$ cd linux
$ make odroidxu3_defconfig
$ make -j 8 zImage dtbs modules
$ sudo cp arch/arm/boot/zImage \
arch/arm/boot/dts/*.dtb /media/boot
$ sudo make modules_install
$ sudo make firmware_install
```

ODROID-XU4

```
bit.ly/1J9ZVn1
```

```
$ git clone --depth 1 --single-branch \
-b odroidxu4-v4.2 \
https://github.com/tobetter/linux
$ cd linux
$ make odroidxu4_defconfig
$ make -j 8 zImage dtbs modules
$ sudo cp arch/arm/boot/zImage \
arch/arm/boot/dts/*.dtb /media/boot
$ sudo make modules_install
$ sudo make firmware_install
```

Building a test kernel

Copy the contents of /media/boot to a new directory on your boot partition, such as /media/boot/backup. If your test kernel has the same version as your current one, you should define a naming extension. For example, appending "-dev" with CONFIG_LOCALVERSION="-dev" in the kernel configuration .config in order to prevent the current kernel modules from being overwritten. Copy the kernel image and the device tree binary to /media/boot/test instead of /media/boot. You can tweak the file /media/boot/boot.ini and modify the path for loading the kernel and the device tree binary. Here is an example boot.ini for the ODROID-C1:

```
setenv prefix '/test/'
...
fatload mmc 0:1 0x21000000 ${prefix}uImage
fatload mmc 0:1 0x22000000 uInitrd
fatload mmc 0:1 0x21800000 ${prefix}meson8b_odroidc.
dtb
...
```

If you want to switch back to your backup kernel image, you change the u-boot variable prefix to /backup/. If you have access to u-boot via serial console, you may define a u-boot macro, which loads the kernel image and the device tree binary from /media/boot/test. Please refer to bit.ly/1JBhhnQ for more details. Don't forget to run the following command before building another kernel:

```
$ make clean
```

If you have comments, questions, or suggestions, please visit the original post at http://bit.ly/1NVRprY

UNIVERSAL IMAGE INSTALLER

by @loboris

I wrote a new universal installer which can install Android, Linux or both as a dual boot system, from SD card and/or USB drive. It may be downloaded from http://bit.ly/1khGRrg, and the source code is available at http://bit.ly/1khGVHB.

Features

- Based on Linux initramfs
- Interactive installer
- User can set desired partition sizes and installation destination
- Can install to SD card or to eMMC
- Uses standard Android upgrade.zip and Linux image files as sources
- Android installation source (update.zip) can be placed on SD card or USB drive
- Linux installation source must be placed on USB drive
- Installation sources must be placed on the first USB drive partition
- Supports dual boot installation (Android & Linux)
- Tested with all Android and Linux versions for ODROID-XU3/XU4

Usage

First, download the pre-built universal installer SD card image, and use md5 to check the integrity of the downloaded files. Unpack the xz archive, which will create an image file. The universal_install_small.img is a 200MB image, and the universal_install.img is a 2GB image. Write the image to an SD card using the dd command under Linux, or image writing software under Windows. You can then expand the FAT partition on the SD card to fit the card's size if you want, but be careful not to change the partition start sector.

Next, copy your installation sources, which is the update.zip file for Android and the .img file for Linux, to the first partition of your USB drive. Rename the Linux installation to linux.img. If you want to install only Android, you can copy update.zip directly to the SD card without using USB drive. Set the ODROID boot switch to boot from SD card, connect your USB drive to the ODROID, insert the SD card and power it on. Follow the instructions to select the desired partition sizes and installation destination (SD card or eMMC).

If you are interested in how it works, download universal_install_source.tar.gz from the same directory. You can create a bootable universal installer SD card, or simply analyze the install scripts to learn more about the ODROID boot process and initramfs. Dual boot functions the same as descibed the article at http://bit.ly/1j9r6TG. Basically, it presents the boot menu to select the OS to boot.

When installing a single OS, you have full freedom to select partition sizes, and you can install to eMMC without removing it from the board. Besides that, the CM-12.1 Android 5.1.1 Lollipop and CM-12.1 Android TV 5.1.1 Lollipop do not have the installation image. Many more options will be added soon, such as backup and restore functions. If you have questions, comments, or suggestions, please visit the original post at http://bit.ly/1PbDhMb.

CPU AND FAN CONTROL

TAME YOUR XU3 AND XU4 HEAT OUTPUT WHEN YOU DON'T NEED FULL OCTA-CORE POWER

by Adrian Popa

If you'd like to improve the fan noise of your ODROID-XU3 or ODROID-XU4, you have a few options. You can change the fan (http://bit.ly/1BeKEqw), or mount an awesome heatsink or case (http://bit.ly/1T28KQ3). However, if you are using your XU3/4 mostly as a server, there's another way to minimize the sound without changing any hardware.

If you update the processor's governor setting and limit the maximum frequency to around 600MHz, the ODROID will rarely exceed 65C, even under heavy load. To change the frequency and governors, you need to edit special pseudo-files inside the /proc directory. To make the job easier for myself, I wrote a small Perl script that can set and list the current values called "odroid-cpu-control". It is available for download at http://bit.ly/1IUZ0qz, and the support topic is at http://bit.ly/1RSrjb6.

The script can list the current frequencies and governors, and set new ones, which requires root privileges. For instance, to list the current minimum and maximum frequencies and governor, you can type the following command, as shown below.

```
$ odroid-cpu-control -l
```

```
adrianp@procyon:~$ odroid-cpu-control -l
CPU0: governor powersave        current 200.00MHz       min 200.00MHz [200.00MHz]       max 600.00MHz [1.40GHz]
CPU1: governor powersave        current 200.00MHz       min 200.00MHz [200.00MHz]       max 600.00MHz [1.40GHz]
CPU2: governor powersave        current 200.00MHz       min 200.00MHz [200.00MHz]       max 600.00MHz [1.40GHz]
CPU3: governor powersave        current 200.00MHz       min 200.00MHz [200.00MHz]       max 600.00MHz [1.40GHz]
CPU4: governor powersave        current 200.00MHz       min 200.00MHz [200.00MHz]       max 600.00MHz [2.00GHz]
CPU5: governor powersave        current 200.00MHz       min 200.00MHz [200.00MHz]       max 600.00MHz [2.00GHz]
CPU6: governor powersave        current 200.00MHz       min 200.00MHz [200.00MHz]       max 600.00MHz [2.00GHz]
CPU7: governor powersave        current 200.00MHz       min 200.00MHz [200.00MHz]       max 600.00MHz [2.00GHz]
```

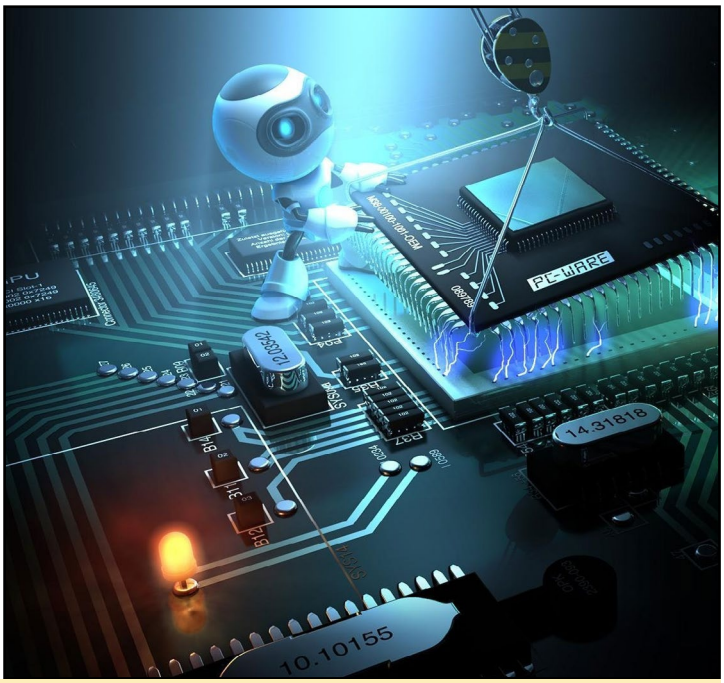
Listing current values

You can be more specific with what you want to list by specifying the CPU core(s) and the parameter(s) that you want listed. For instance, to display current frequency and max frequency for cores 1, 4, 5 and 6, you can type the following, as shown in Figure 2:

```
$ odroid-cpu-control -l -f -M -c 1,4-6
```

```
adrianp@procyon:~$ odroid-cpu-control -l -f -M -c 1,4-6
CPU1: current 200.00MHz max 600.00MHz [1.40GHz]
CPU4: current 200.00MHz max 600.00MHz [2.00GHz]
CPU5: current 200.00MHz max 600.00MHz [2.00GHz]
CPU6: current 200.00MHz max 600.00MHz [2.00GHz]
```

Listing selective attributes



Sometimes you need less power than the ODROID's full capabilities

The parameter functions are explained if you run the script without arguments. In summary, -l handles listing, -f shows current frequency, -m handles minimum frequency, -M handles maximum frequency, -g is the governor and -c specifies the cores you want to operate on. The syntax for the cores is flexible. You can either separate them by comma without spaces, or use a dash for ranges. If you omit the -c argument, all CPU cores will be selected by default.

The script also provides a clever way of keeping an eye on the current parameters, similarly to how the command "top" operates. If you add -i 1 to the list syntax, then the data will refresh every second. This is mostly useful when looking at the current frequency that the cores operate on when doing CPU intensive tasks.

To make changes to the CPU configuration, you obviously need to run the script as the root user using sudo. The syntax changes slightly: you replace -l by -s (for set). The -m, -M and -g options use additional parameters. To change the minimum and maximum frequency, you need to specify the new frequency in MHz or GHz with the suffix M or G. The available governors are listed when running "odroid-cpu-control -h". For example:

```
$ sudo odroid-cpu-control -s -m 300M -M 1.2G -g ondemand -c 0,4
```

```
adrianp@procyon:~$ sudo odroid-cpu-control -s -m 300M -M 1.2G -g ondemand -c 0,4
CPU0: governor powersave -> ondemand
CPU0: min 200.00MHz [200.00MHz] -> 300.00MHz [200.00MHz]
CPU0: max 600.00MHz [1.40GHz] -> 1.20GHz [1.40GHz]
CPU4: governor powersave -> ondemand
CPU4: min 200.00MHz [200.00MHz] -> 300.00MHz [200.00MHz]
CPU4: max 600.00MHz [2.00GHz] -> 1.20GHz [2.00GHz]
```

Setting min/max/governor

In the command above, we set the minimum frequency to 300MHz and the maximum frequency to 1.2GHz for cores 0 and 4, while switching the governor to ondemand. Note that on the XU3 and XU4 platforms, the big.LITTLE architecture sets the same parameters for cores 0-3 and 4-7. For example, changing something on core 0 will change the same thing on core 1, 2 and 3 as well. This means that the command above effectively changes the settings on all of the cores. If your values for frequency are invalid, such as going over the maximum frequency supported by that core, they will be rounded to the nearest valid value by limiting it to the maximum frequency). Because of this rounding, the following command is perfectly valid, although it won't overclock your CPU:

```
$ sudo odroid-cpu-control -s -m 10M -M 4.5G -c 0,4
```

```
adrianp@procyon:~$ sudo odroid-cpu-control -s -m 10M -M 4.5G -c 0,4
CPU0: min 200.00MHz [200.00MHz] -> 200.00MHz [200.00MHz]
CPU0: max 1.40GHz [1.40GHz] -> 1.40GHz [1.40GHz]
CPU4: min 200.00MHz [200.00MHz] -> 200.00MHz [200.00MHz]
CPU4: max 2.00GHz [2.00GHz] -> 2.00GHz [2.00GHz]
```

Out of bounds parameter handling

You might be wondering how the above command might help you. First, you'll need to work out which CPU governor is best for your use case. In my opinion, the default governor of "performance" is wasteful because it keeps the frequency high and causes the system temperature to rise. Other governors increase or lower the core frequency based on workload by using various algorithms, with the "powersave" option trying to keep all frequencies to a minimum.

You can combine the governor and minimum/maximum frequency settings to give your system the best balance between responsiveness and temperature. It might be possible to find a combination where you can use the XU3/4 with Kodi without hearing the fan. Other platforms may benefit from lowering CPU frequency to reduce heat, or gain increased battery life if the system is run on battery.

There will be times when one setting is not optimal. In that case, you can use the "cron" application to set various profiles at different times, or simply switch settings on the fly using the command line shown above. For example, you could limit the maximum frequency at night and turn it up in the morning.

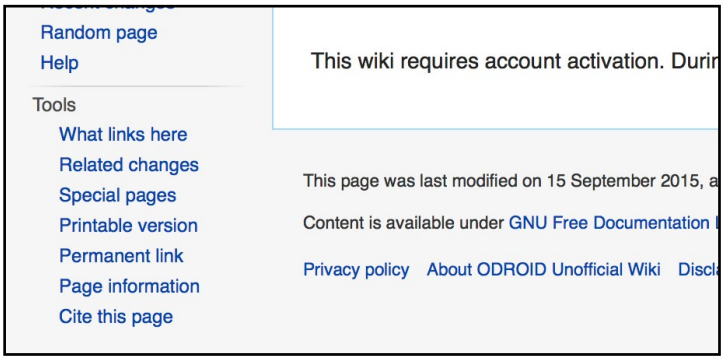
COMMUNITY WIKI

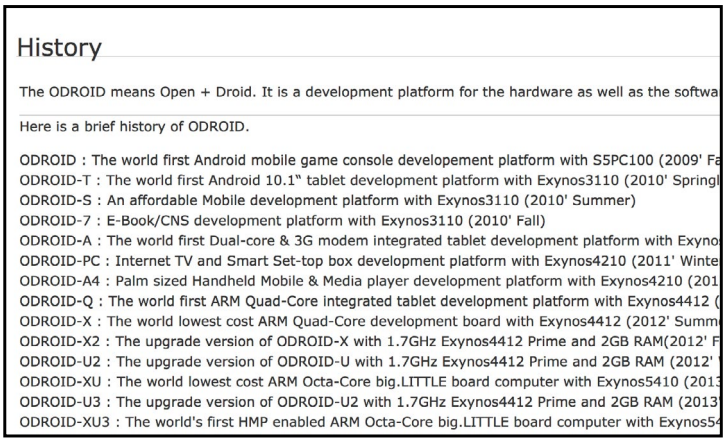
CONTRIBUTE TO THE EXPANDING ODROID KNOWLEDGE BASE

by Rob Roy

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

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





ODROID MAGAZINE WEBSITE

A NEW DESIGN FOR OUR THIRD YEAR

by Rob Roy

Since we're celebrating the beginning of our third year of publication, ODROID Magazine has built a new website! Check it out at `http://magazine.odroid.com`. It offers several improvements and features:

Search form to quickly find articles

Master Table of Contents with links to individual articles
`magazine.odroid.com/articles`

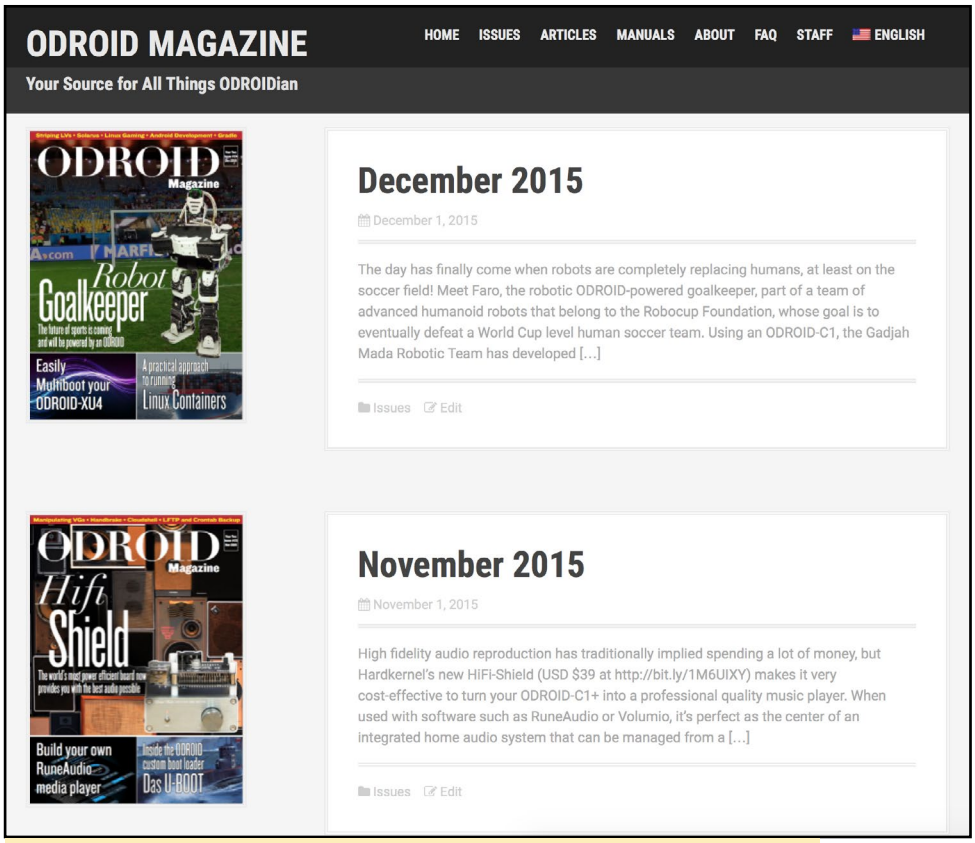
User manuals section

Donation section
`magazine.odroid.com/about`

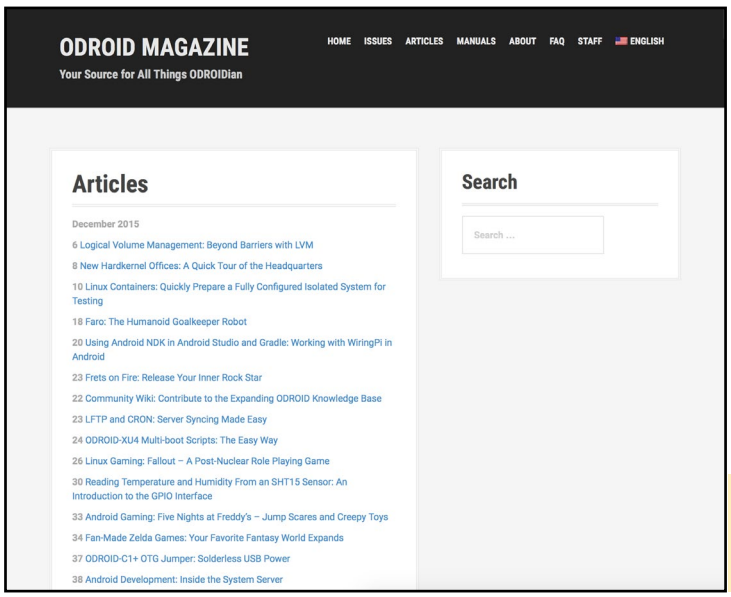
Quick access to individual issues by typing magazine.odroid.com/YYYYMM
`magazine.odroid.com/201512`

- Compatible with desktop and mobile

- Available in Spanish



Home page of the new ODROID Magazine website showing the latest issues



The new site has a searchable table of contents linking every article from every issue

We will be continuing to update the layout and look of the site over the next few months, so watch for more improvements. For comments, questions, or suggestions, please visit the original post at `http://bit.ly/1k5x8Ea`.

ANDROID 6.0 MARSHMALLOW FOR ODROID-XU4

THE NEWEST ANDROID FOR YOUR NEWEST ODROID

by @voodik

Forum user @voodik recently released a version of Android 6.0.1 Cyanogenmod 13.0 for the ODROID-XU4. The features include Linux kernel version 3.10.9, OpenGL ES 1.1 / 2.0 / 3.0 and OpenCL 1.1 EP. It allows up to 8 users, a portable WiFi hotspot, and HDMI-CEC support.

This month, Android Marshmallow began its international release to smartphones, and offers several improvements over Lollipop, including visual updates, intelligent battery management, and more robust application security features. @voodik has also included Google Play in his build, so you don't need to install it separately.

Installation

The SD card or eMMC module first needs to be prepared using one of the self-installation images, available at `http://bit.ly/1QO9Wcn`. More information about flashing images may be found on the Hardkernel Wiki at `http://bit.ly/1Vk9u4o`.

When using an SD card, flash the sd_installer image to the card, then insert the SD card into the ODROID-XU4. When using an eMMC module, the self-installation image called sd2emmc_installer should be flashed to a temporary SD card, after which both the SD card and the eMMC module should be attached to the ODROID-XU4.

With the boot media hardware switch set to "SD card," power on the ODROID-XU4 to install the operating system. The blue LED will remain solid during the process and the fan will run. Once the installation has been completed, the ODROID will power off automatically. If using the eMMC module, switch the boot media hardware switch to "eMMC." Then, power on the ODROID and enjoy Android Marshmallow!

Known issues

The release is still a work in progress, and the Bluetooth and USB-3G support is still pending. The Google search bar needs to be disabled from the Launcher settings and re-added from the widgets in order to display properly, and MTP should be disabled, then re-enabled using the Developer Options -> Select USB Configuration menu.

Tips

To get WiFi working, set the correct module name in build.prop. For example, to use the RealTek 8192cu default module, add or update the following line:

```
wlan.modname=8192cu
```

For the Realtek 8188eu module, use the following line:

```
wlan.modname=8188eu
```

The Ralink RT33XX/RT35XX/RT53XX/RT55XX module would be specified like this:

```
wlan.modname=rt2800usb
```

To enable USB GPS, set the correct TTY and speed in the build.prop file:

```
ro.kernel.android.gps=ttyACM0
ro.kernel.android.gps.speed=9600
```

For comments, questions, or suggestions, please visit the original post at `http://bit.ly/1YEt9BG`.

UNIVERSAL MOTION JOYPAD

ARE YOU READY TO DRIVE A RACE CAR?

by John Lee and Charles Park

The Universal Motion Joypad is Hardkernel's new game controller that is similar to the steering wheel available for the Nintendo Wii. It is built using the USB-IOBOARD and an auxiliary motion sensor, and allows you to play many games available for mobile phone that would normally use the phone's gyroscope. It's available for purchase at the Hardkernel store (`http://bit.ly/1Sbe46q`) for USD$40, and provides many hours of gaming fun!

Figure 1 - Closeup of the Universal Motion Joypad

Features

The Joypad is customizable by adjusting the boot.ini file of the ODROID without the need to recompile any packages. The printed circuit board (PCB) is designed as a universal board. Therefore, if users can do soldering by themselves, they can attach switches or input devices.

Components

- Accelerometer Sensor
- Universal Board
- Round Key Button 10 EA
- Status LED 2EA
- 8-bit MCU
- USB Interface

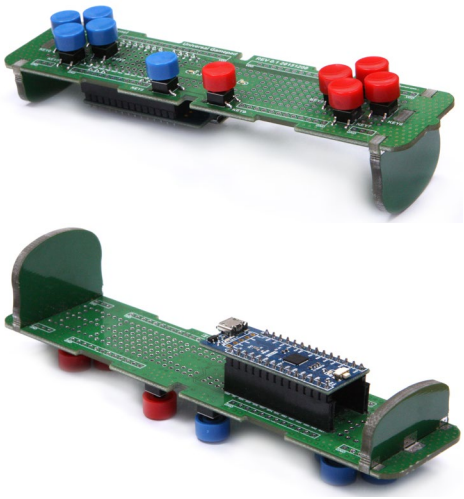


Figure 2a and 2b - Top and bottom view of the Joypad PCB

Design

The Universal Motion Joypad is composed of a simple hardware structure. The PIC18F45K50 MCU in the USB-IOBOARD reads the keypad switches,BMA150 accelerometer sensor, and controls the status LEDs.

The USB-IOBOARD uses the PIC18F45K50 microcontroller unit (MCU) made by Microchip, which enables the GPIO, serial and timer functions. As shown in Figure 4, the GPIO and I2C are used to control Universal Motion Joypad. The USB-IOBOARD reads the signal of key switches through the GPIO ports. The switches normally show a HIGH state, and reports a LOW state when pressed.

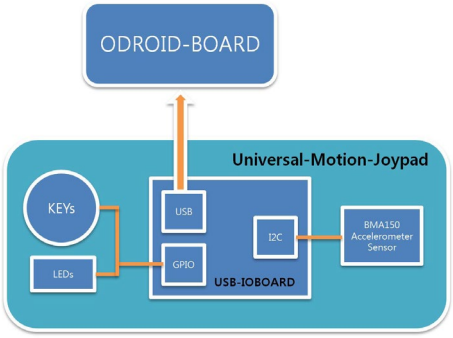


Figure 3 - Joypad Block Diagram

The accelerometer sensor is connected to the I2C bus. When the MCU boots up, the corresponding registers initialized. Each keypad and accelerometer sensor frequently reads the values at regular intervals using the MCU's timer, and the values are stored as 10-byte packets and transmitted to the ODROID via USB interface.

Each 10-byte packet contains a header and tail taking 1 byte each, along with 8 data bytes. The data consists of 6 bytes of acceleration sensor data, and 2 bytes of keypad data. The PCB labels KEY0 - KEY9 values are mapped to the cor-

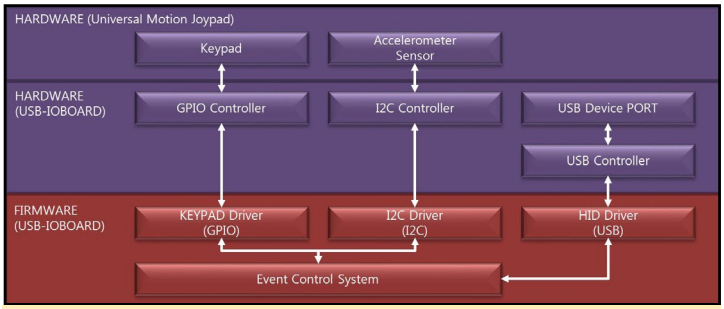


Figure 4 - Joypad Interface Diagram

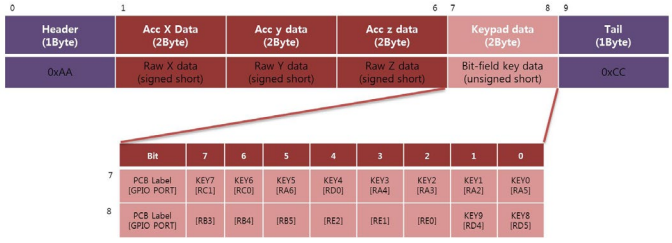


Figure 5 - Joypad data packet byte mappings

responding GPIO as shown in Figure 5, with the option to use additional input ports. The GPIO port and key mapping are easily configured using the boot.ini file of the ODROID.

Android framework

As shown in Figure 6, the application framework is an application uses the sensor framework to obtain the sensor's data. It communicates with the C++ layer through the Java native interface (JNI). The sensor library middle layer mainly consists of the sensor manager, sensor service and sensor hardware abstraction layer.

The input subsystem is a generic Linux framework for all input devices like keyboard, mouse, and touchscreen, which defines a standard set of events. It interfaces to the user space through the /sys/class/input interface. The event device (Evdev) provides a generic way for input device events to be accessible under /dev/input/eventX. The sensor interface driver communicates with the USBIO Board via USB bus.

Beach Buggy Racing

One of our favorite Android racing games is Beach Buggy Racing. It's a Mario Kart style game that features amazing

Figure 6 - Android framework diagram for the Joypad drivers

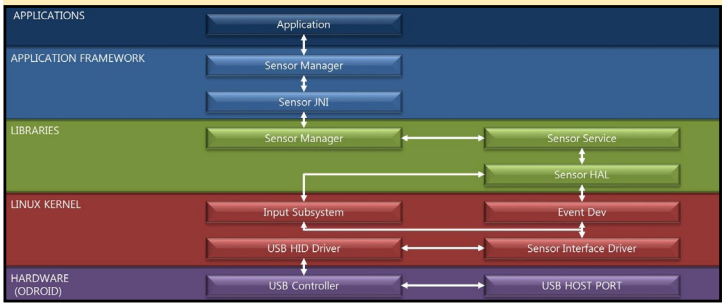


Figure 7 - Beach Buggy Racing

graphics, smooth gameplay and up to 4 player action. The game is compatible with many USB joysticks and HDMI touchscreens, but is much more fun and easy to drive using the Universal Motion Joypad!

To use the Universal Motion Joypad, it's first necessary to configure the parameters of "acc_orientation", "button_map" and "usbhid.quirks" in the boot.ini file on the Android FAT32 partition. After saving the file, reboot the ODROID for the new parameters to take effect.

```
setenv orientation "3"
setenv bts "0:M:L,1:M:U,2:M:R,3:M:D,4:T:71:203,5:K:10
2,6:T:1214:212,7:T:1253:43,8:M:l,9:M:r"

setenv bootargs     "fb_x_res=${fb_x_res} fb_y_
res=${fb_y_res} hdmi_phy_res=${hdmi_phy_res}
edid=${edid} hpd=${hpd} led_blink=${led_blink} acc_
orientation=${orientation} button_map=${bts} usbhid.
quirks=0x04d8:0x003f:0x0004"
…
```

These changes send three parameters to the kernel for key mapping. Note that some other games may need a different orientation set in the acc_orientation parameter. The button_map sets the key mapping of each button. Each key can be mapped to a mouse event, a touchscreen event or a keyboard event. The usbhid.quirks parameter forces the USB-IO board to work as a generic input driver instead of the standard HID device. A detailed explanation of the acc_orientation and button_map parameters are shown in Figures 7a and 7b.

Button mappings

The mapping of the keypad's operation can be set by using the "button_map" boot argument. The key label's number is used as the first parameter. The second parameter specifies the operation, with the third and fourth parameters setting the corresponding values of the operation. The buttons can

KEY	type	value1	value2
KEY Number (PCB Layout)	M	U : Move up mouse cursor D : Move down mouse cursor L : Move left mouse cursor R : Move right mouse cursor l : click mouse left button m : click mouse middle button r : click mouse right button	
	T	Touch X position	Touch Y position
	K	Key code	

Figure 8a - Button mapping table

Accelerometer sensor orientation				
Sensor PCB	2	3	0	1
PCB Sensor	7	4	5	6

Figure 8b - Accelerometer mapping table

also be mapped to mouse functionality or touchscreen points instead of normal key inputs.

```
button_map=[KEY]:[type]:[value1]
:[value2]
```

For example, to map the KEY0 input to a mouse left click and the KEY1 to a touching coordinates x=200, y=200, the parameters would be:

```
button_map=0:M:l,1:T:1200:200
```

You can obtain the coordinates for a touch input by enabling the "Settings" -> "Developer options" -> "Pointer location" setting on Android.

Accelerometer mappings

The "acc_orientation" argument can be used to customize the Joypad operation in any direction. It is changed by setting the parameter of the accelerometer sensor:

```
acc_orientation=<Orientation
value>
```

For example, if you want to read the value of rotating the Joypad 90 degrees clockwise, you would adjust the acc_orientation value to 2:

```
acc_orientation=2
```

Operating system support

To use the Joypad, you first need to update the Android operating system to the latest version:

- ODROID-XU3/XU4 Android version 4.4.4 (v3.3), release date Dec 22, 2015, kernel version 3.10.9: http://bit.ly/1Ossuyo
- ODROID-C1 Android version 4.4.2 (v2.1), release date Dec 11, 2015, kernel version 3.10.33: http://bit.ly/1JB0jWs

Key mapping for Beach Buggy Racing

So that you can get started quickly with playing Beach Buggy Racing, here is the joypad key mapping for the game that would be added to the boot.ini file:

```
setenv bts "0:M:L,1:M:U,2:M:R,3:M
:D,4:T:71:203,5:K:102,6:T:1214:21
2,7:T:1253:43,8:M:l,9:M:r"
```

This parameter string sets the inputs to the following values:

```
KEY0 = Mouse Left direction
KEY1 = Mouse Up direction
KEY2 = Mouse Right direction
KEY3 = Mouse Down direction
KEY4 = Touch Click (Left up side
item click : X = 71, Y = 203)
KEY5 = Key (KEY Home : keycode is
102)
KEY6 = Touch Click (Right up side
item click : X = 1214, Y = 212)
KEY7 = Touch Click (Game Pause
click : X = 1253, Y = 43)
KEY8 = Mouse Left button
KEY9 = Mouse Right button
```

The X-Y coordinates of the touchscreen are tuned for an HDMI screen resolution of 1280x720. If you are using a different resolution such as 1920x1080, the coordinates must be changed. As previously mentioned, the touch coordinates may be obtained through the "Pointer Location" developer setting. Make sure to reboot the ODROID after modifying the boot.ini file.

Hardware overview

- Micro Electro Mechanical System (MEMS) Accelerometer Sensor
- BMA150 Sensor electronic compass with three-axis magnetic field sensor and three-axis accelerometer

Source code

You can learn how to implement the sensor interface HAL, and find out what was changed in the source code in order to add the drivers to Android, by reviewing the following GitHub commits:

ODROID-XU3/4

libsensor sensor library
http://bit.ly/1MATKDh

sensor.odroidc.so library
http://bit.ly/1YIFgIB

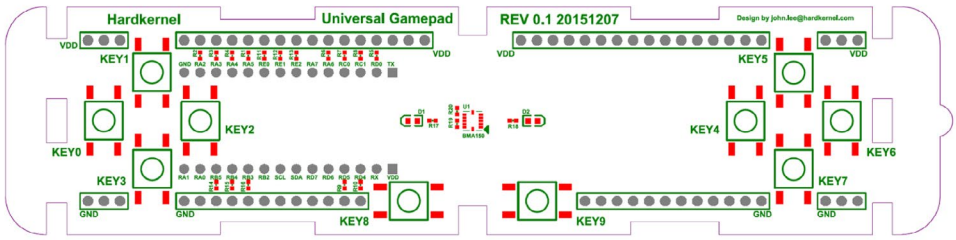
usbio-keypad.idc
http://bit.ly/1PpgYB0

USBIO sensor board driver
http://bit.ly/1OmbLX9

kernel configuration file
http://bit.ly/1Ost5jG

USBIO key pad driver
http://bit.ly/1J9PRur

Figure 9 - Joypad PCB diagram



ODROID-CI

libsensor sensor library
http://bit.ly/1MATKDh

sensor.odroidc.so library
http://bit.ly/1kkTTEq

usbio-keypad.idc
http://bit.ly/1TlxlQ1

USBIO sensor board driver
http://bit.ly/1mbuGy3

kernel configuration file
http://bit.ly/1R1wAfJ

USBIO key pad driver
http://bit.ly/1mbuLSq

USB-IOBOARD

MCU Firmware source tree
http://bit.ly/1R1wG75

MCU firmware download
http://bit.ly/1QSV1NR

Demo video

Check out the Universal Motion Joypad in action at http://youtu.be/xSlPJzXEsIo.

In Beach Buggy Racing, you can drive as a rabbit and fire carrots at your nearest competitor (no kidding, you really can!)

RESPECT YOUR COWORKER'S JOB

PLEASE, DON'T TOUCH ANYTHING SHOWS THAT IGNORANCE IS BLISSFUL FUN.

by Bruno Doiche

You're left in charge of a desk and a big red button while your co-worker takes a bathroom break, having instructed you not to touch anything. So what's your first instinct? To touch something, of course!

This is the premise of Please Don't Touch Anything: you are just sitting in a desk controlling a mysterious apparatus that has multiple logical and illogical controls capable of ending civilization as we know it.

With plenty of different endings, amazing creativity, you will spend hours trying the possible and the impossible combinations of actions that this game has to offer!

Please Don't Touch Anything is available for download from the Google Play Store at the following link:

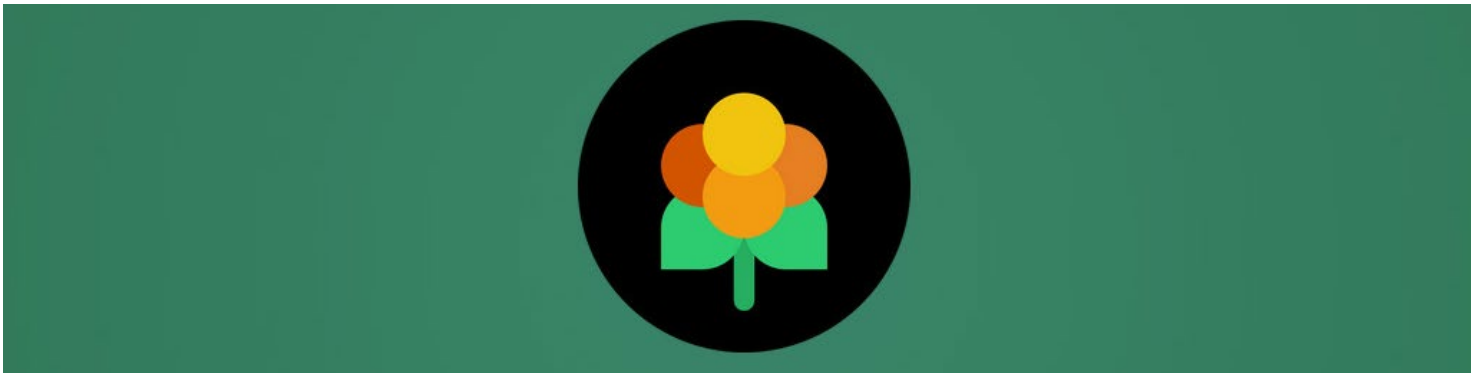
https://play.google.com/store/apps/details?id=com.fourquarters.PleaseDontTouchAnything

Ranging from the a little obvious to the absolute crazy, this games will devour hours of your time, make the best of it!

LAKKA FOR ODROID-XU4

THE ULTIMATE GAMING SYSTEM

edited by Andrew Ruggeri



Lakka is a lightweight Linux distribution that transforms a small computer into a full-blown game console. The OS, which can run on both the ODROID-XU4 and ODROID-C1, features a multitude of different emulators, known as "cores", as well as an attractive user interface. The Lakka developers have recently announced the release of a new major version of Lakka. This new version is still based on OpenELEC 5, but ships with the latest RetroArch, with a lot of changes on the graphical interface. The download links and the installation guide have been updated to reflect the new way of launching games.

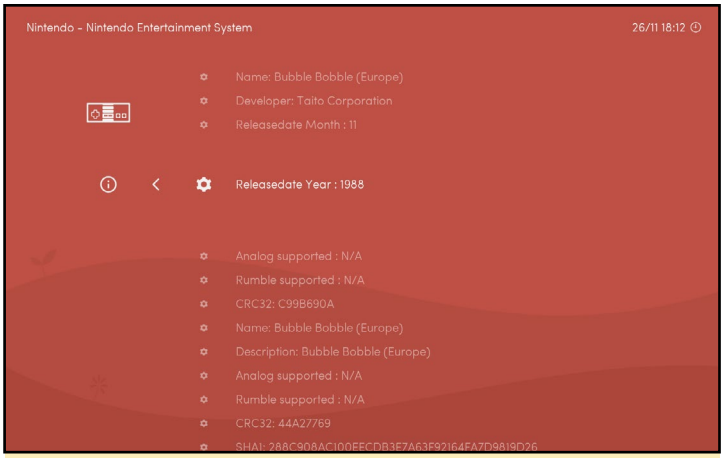


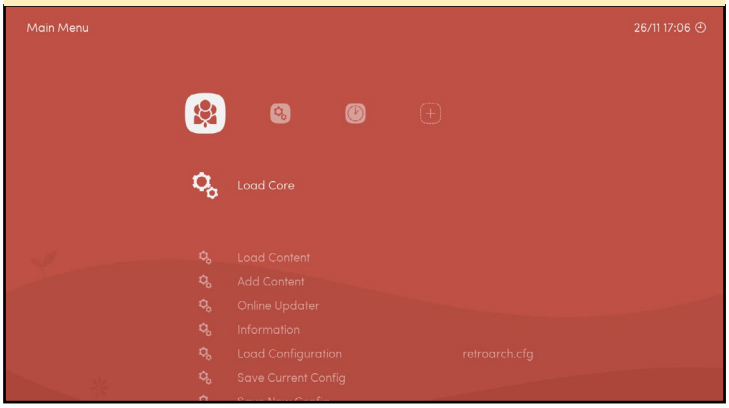
Figure 2 - Lakka now includes a game database

Features

When booting a fresh installation of Lakka, you will start with 4 tabs, as shown in Figure 1.

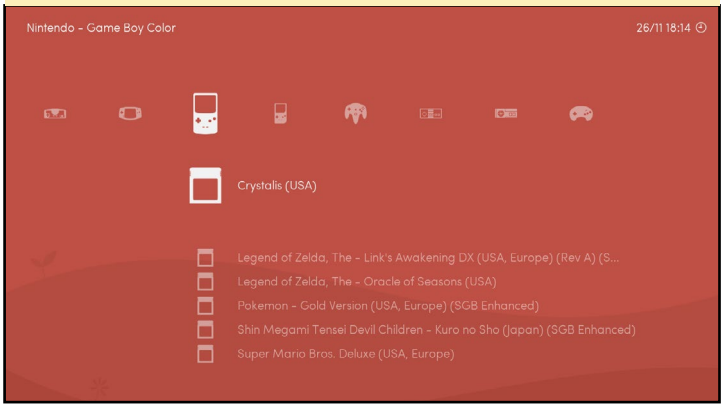
- The RetroArch Menu, to launch ROMs manually, and shutdown the OS
- The Settings tab, to tweak your Lakka instance
- The History tab, to browse the games you played recently
- The + tab, to add new custom tabs
- Database, Scanning and Playlists

Figure 1 - Lakka tabs



The new RetroArch included in Lakka introduces a game database. This database is nothing more than the DAT files from the famous No-Intro initiative `bit.ly/1RKJT5L`, converted to binary for performance reasons. It contains metadata about the games, like the publisher and the release date. With this, it becomes easy to browse games made by the same team. This database also contains the checksums of ROMs, and RetroArch uses these checksums to match your ROMs against the

Figure 3 - Lakka has a new scanning system



database entries.

Since users were constantly complaining about the wrong game sorting in the previous versions of Lakka, we implemented a scanning system that will check all your ROMs against the database and generate one playlist by system. This is why we recommend that you use the No-Intro romsets from now, since they will scan properly. We ensure that adding custom playlists that doesn't refer to a game system or contains ROMs not part of the no-intro sets will work.

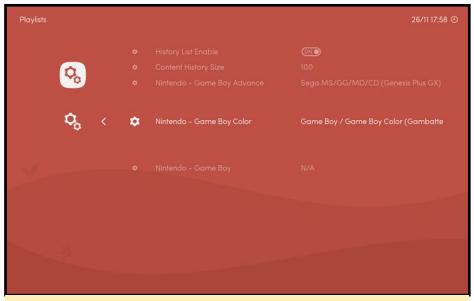


Figure 4 - Lakka assigns a default core to each playlist

A default libretro core is assigned to each playlist. You can customize this choice in the Playlist Settings. You can also completely remove that association by pressing START. RetroArch will then let you choose the core on a per-ROM basis.

Dynamic backgrounds and boxarts

RetroArch developers recently added non-blocking IO that allows us to load images without causing lag in the interface. With this, we can release a first version of two popular features: dynamic backgrounds and boxarts.

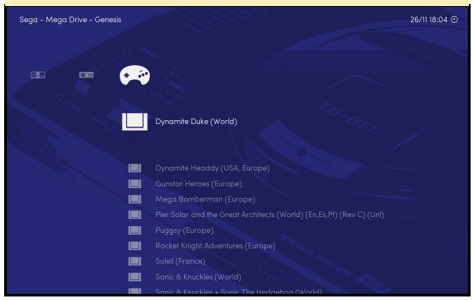
With dynamic backgrounds, you can associate a custom background image with each entries of the horizontal list:

- **Enable the feature in the Menu Settings**
- **Set your Dynamic Background Dir in the Directory Settings to point to a folder containing the images.**

Name the images after the titles that appear at the top left corner in our interface. The format needs to be RGBA PNG. This feature will lag on weak hardware, but will work well for PC users. Three ready-to-use packs are provided in this release under the /usr/share/retroarch-assets/wallpapers/ directory.

Boxarts works exactly the same, except that you have to follow a directory structure in order to avoid naming conflicts. For example, "boxarts/Nintendo - Game Boy/Named_Snaps/Tetris (World).png". Images need to be in RGBA PNG format, and have to be named according the entries in the vertical list. We don't provide boxart packs yet, but the project No Intro Screenshot Reloaded (`http://bit.ly/109Vuoi`) provide some ready-to-use screenshot packs.

Figure 5 - Custom background images can be selected



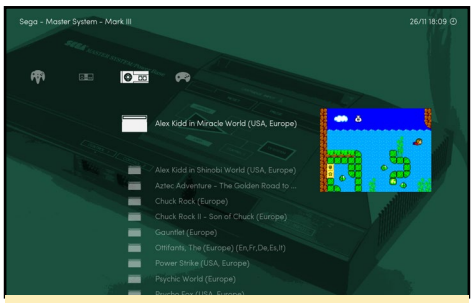


Figure 6 - Box art can also be displayed for games

DRM/KMS

A big bug with Radeon GPU was preventing us from providing DRM/KMS builds of Lakka for PCs. @lugaidster on RetroArch issue tracker found a workaround that we can use in Lakka, allowing us to remove the huge X11 dependency again. However, some users reported too many incompatibilities and poor performance with the DRM/KMS builds. We still believe that being X11-free is the future, and that vendors will soon release proprietary drivers that are compatible with DRM/KMS. In the meantime, I may provide unofficial X11 builds somewhere.

Bug fixes

The 55 FPS bug that was affecting the ODROID-XU3 and XU4 is finally gone. It was not a display bug. We just needed to set the audio rate to 44100 instead of the default 48000. Special thanks to @maister who found the root cause of the problem. We are very happy to provide a stable release for the XU3 and XU4 after all this time.

Missing features

Scanning is a work in progress. The following systems still need to be implemented:

- **Arcade scanning**
- **PCE-CD scanning**
- **Sega CD scanning**

Upgrading

The configuration files of RetroArch and the directory structure of the storage partition changed a lot. Therefore, we recommend not to use the regular upgrade path, and to do a fresh install this time. Don't forget to backup your games and saves. If you have questions or comments, please visit the original post at `http://bit.ly/1PSklD7`.

LINUX GAMING

STRATEGY GAMES ON THE ODROID PART I

by Tobias Schaaf

I was always a PC gamer, and one of my favorite genre is widely available on PCs, but rarely to be found on consoles: strategy games! I really love them, and even as a child, I was fascinated by these types of games. On our old Amiga, starting with Dune 2, this was a genre that managed to keep me busy for hours and hours. The aspect of planning out your strategy, building up an army, and crushing your enemy appealed to me. I have very many fond memories of awesome multiplayer sessions with my friends playing StarCraft and Total Annihilation. Even today, I really enjoy these kind of games. I'm looking forward to buying the trilogy of StarCraft 2 soon. Since there are so many strategy games, and it's really hard to decide which are the best ones, I'll concentrate on good old DOS games, which can be run on the ODROID platform using DOSBox.

Figure 1 - Panzer General, a 2nd World War simulation where you lead armies into battles. Every loss is a pre-calculated necessity in this game

Figure 2 - Settlers 2 one of these games that can keep you busy for many hours just planning on how to structure all your good routes

Subgenres

There are many different subgenres of strategy games available. Some are tactical strategy games like History Line 1914-1918, Battle Isle, Gettysburg, and Panzer General, where you have a limited amount of units and have to plan your moves to reach your goal, drive back the enemy, or conquer a point of interest. You have limited options to increase the number of units, such as calling backup, or investing what little money you have to buy replacement units. You generally do not throw unit after unit against an enemy until you've won the battle. Instead, you plan every move, and know that every unit counts.

Another subgenre is the simulation strategy game, where you have to manage far more than your troops and your base. Typically, you have to adjust and distribute resources, consider diplomacy, and organize multiple sites and multiple conflicts at the same time. The best known franchise of this type is probably the Civilization series, but there are many more, such as Imperium Galactica, Master of Orion, Fragile Alliance, the Settlers series, and the Anno series.

The last subgenre I want to discuss is the base-building kind of strategy game, including games like the afore-mentioned Dune 2, StarCraft, the famous Command and Conquer series, and many more. These are my favorite kinds of strategy games: where you build up a base, amass a large army, and try to crush your enemy, all while building your defenses up to hold back whatever the enemy is throwing at you.

There are more sub-genres of strategy games, like round-based strategy games such as my favorite XCOM series, or the Heroes of Might and Magic series, but for now I want to limit the number and genres of games to a few gems. Since I mostly enjoy the base and army building kind of games, I will concentrate on those.

Preparation

Since the games I want to focus on are all for DOS, I use my trusty DOS-

Box emulator to play them. Because I have a great working version of DOSBox in my repository, it should be fairly easy for you to get these games to run in DOSBox. First, create a folder for storing games, as well as a subfolder called CDs for storing ISO files, then install DOSBox:

```
$ mkdir -p DOS/CDs
$ sudo apt-get install dosbox-
odroid
```

Start DOSBox once to create the default config file, but then exit it right away. Open /home/odroid/.dosbox/dosbox-SVN.conf in a text editor and change the following lines, after which you can restart DOSBox from the Applications menu:

```
[sdl]
fullscreen=true
fullresolution=desktop
windowresolution=1024x768
output=overlay
[dosbox]
memsize=31
[render]
 frameskip=3
aspect=true
[cpu]
core=dynamic
cycles=auto
```

For convenience, you should also add the following lines to the end of your DOSBox configuration file found in /home/odroid/.dosbox/, replacing MyISO.iso with the name of your ISO file:

```
[autoexec]
mount c: /home/odroid/DOS -free-
size 1024
imgmount d: /home/odroid/CDs/
MyISO.iso -t iso
c:
```

Once the emulator has launched, the folder DOS will automatically be mounted as the C: drive and MyISO.iso will be mounted as D: as a CD-ROM drive.

Note that I added the option -freesize 1024 for mounting the drive C: in DOSBox, which is needed for games that check the size of available space on the hard drive. Without this option, the C: drive is only mounted with less than 300MB reported as free, even though it's the size of your SD card or eMMC module.

Command And Conquer

One of the first games of this genre was Dune 2, which people lovingly call the grandfather of all real-time strategy (RTS) games. It was the first major success after Westwood, where you build a base and units and send them into battle. Whereas Dune 2 played on a distant world with great houses fighting against each other about territory and power, Command and Conquer pulled us back to Earth and let us participate in an-all out world war again, where two different factions, GDI and NOD, try to fight each other.

Figure 3 - Command and Conquer used lots of movies and cut scenes to tell the story it was really good for the time

Command and Conquer was a much larger game than Dune 2 was, with a story to follow where you could identify yourself as the commander. Your superiors directly talked to you in cut scenes, and you could follow the progress of the war in news shows and movies that talked about the impact of your last mission. These features, paired with an awesome

Figure 4 - Building up your base and units was really fun in Command and Conquer. This game is definitely a must have!

battle system, as well as the progression of all of the buildings and units in your arsenal, really made this game one of a kind.

I will always remember the many hours that I played this game, and it's really fun to play it on the ODROID. It brings back good memories of one of the greatest series in gaming history. Command and Conquer 1 and 2 (Red Alert) are both available for DOS and run very nicely on the ODROID platform.

Earth 2140

I really love this game! It's not very well known, and the later games, such as Earth 2150, 2160 and several extension packs, were enjoyable and had really good 3D graphics for the time, but nothing could beat the original. This game was so much more advanced in terms of game graphics than Command and Conquer. It rendered in very high resolution, even under DOS, and the effects were so much better than what

Figure 5 - Smoke, damaged terrain and realistic looking fire, are only a few of the awesome effects of Earth 2140

Command and Conquer could offer.

In Earth 2140, you can play either as the Eurasian Dynasty (ED) or United Civilized States (UCS). Both factions are very different, which also makes the game very unique. As ED, you normally have soldiers and tanks to fight your enemy. As UCS, you use battle mechs and cyborgs, who look very much like the guys from the Terminator movies.

Sadly, the game is lacking a good background story, but you can see where the money went. The interface and entire game had really impressive graphics for a DOS game of its time. What I remember best are the laser tanks of the ED, which I really loved. A group of laser tanks could burn an enemy within seconds. The laser just cut through their armor like a hot knife through butter, and it looks very impressive if there are ten or twenty laser beams combined on a single enemy target.

The UCS, on the other hand, have intimidating mechs which can fire devastating plasma at the enemy, and many of their units have napalm grenades, which do damage over time as well as AOE damage. This is definitely one of the next games on my list to play through again on the ODROID, and one that I'm really looking forward to. If you like strategy games but haven't heard of Earth 2140, grab a copy from GoG.com, and play it, because it's definitely worth it.

Gene Wars

Gene Wars is very unique, and is among the games that I'm currently playing on my ODROID. It's made by Bullfrog, one of my favorite companies from that time. In this game, you can raise different kind of animals, and also cross breed them to achieve your goals. The goals range from getting a certain amount of points, or researching something, to destroying the enemies base.

You have four different kind of workers that you can use: an engineer who will build and upgrade your buildings, a botanic with whom you can grow different kind of plants, a ranger who can help you groom and cross-breed your animals, and a genetic, who will research new species as well as heal your creatures if they get hurt. This game is very fascinating since it combines the standard "build a base with army" objective with some new aspects of growing and cross-breeding species. There is even a botanical aspect, where you have to decide which plants you want to grow in order to produce resources and/or food for your creatures on a specific terrain.

Terrain is another interesting facet of the game. The terrain can actually be changed: hills may be flattened by your builder to make even space for a building, and if a building blows up, it can leave a big crater in the ground, which is very advanced for a DOS game. Ultimately you use your creatures and different workers to solve the tasks that are given to you by an alien species that watches the progress of both you and your enemies. These tasks normally include having a certain type of species at a specific location, which will earn you points and please the aliens, or growing plants at different locations. Often, there will be an enemy player with the same capabilities and goals as you, so you either need to be faster than them, or drive back the enemy, which sometimes can be really hard.

Certain types of creatures can perform only certain types of tasks. For example, the first creatures you will get are mules, which make a great workforce, but are not very good at attacking. They can be used to harvest plants and turn them into goop, which is the basic building material. Later on, you'll find a crab, and can clone them as well through research. Crabs are rather strong compared to the mules, and are excellent fighters. Sadly, they are not very good at regular work. They can fell a tree, but they can not transport them like a mule can.

Later on, you're able to cross-breed these species, which gives you a creature that has stronger defense and attack values than a mule while still being able to transport wares. This kind of mixing can be done with all of the species that you encounter on the planet. There are a total of 5 different species to discover, and they can all be cross-bred with each other. Each cross-breed can have two results, depending on which genes become dominant, which adds up to a total of 25 different creatures you can obtain throughout the game. If you perform well, and the aliens are pleased with you, you will eventually get a monolith which you can use to increase the abilities of your workers and creatures.

The game also comes in many different languages. It does not reinvent the genre, but used some unique aspects of these kind of strategy games. One word of advice is to save often and on different save slots. Since there is no "restart level," you have to load a previously saved game if you fail a level, and can only start a completely new game from the start menu, therefore saved games are necessity.

Figure 6 - Is it a Mule, or is it a Crab? No, its a Mule-o-crab! Umm, I mean a Crab-o-mule

Fragile Alliance

If you are a fan of the Amiga, you may have heard of a game called K240. Fragile Alliance is similar to that game. You are an operative of a mining company called Tetra Corp, and your main goal is to earn credits for your company. To do so, you are supposed to harvest rare

ores from asteroids. You have to build a stable living environment with radiation filters, living quarters for your works, mines to harvest ores, entertainment complexes to keep your workers happy, but also security centers to maintain order on your asteroids. Later on, you can produce your own spacecrafts that have different "slots," which you can equip individually with various weapons or defense equipment. You can even build space stations and larger ships.

Later in the game, you meet other species and have to interact with them through diplomacy or military action. You can colonize new asteroids to get more resources and credits, or conquer them if they are occupied by another species. If you prefer to destroy the enemy before you try to take over an asteroid, there is a wide arsenal of different missiles to do this as well. There are smugglers and traders to sell and buy products, and you can have supervisors helping you to create better asteroids. You can also use spies to sabotage enemy asteroids and productions.

Figure 7 - In Fragile Alliance you settle on asteroids, defend them and build armies and missiles to increase your power and wealth

There are many different tactics to fight enemies, and the weapon arsenal in this game is impressive, especially the missiles. There is even a so-called stasis missile which allows you to freeze an entire asteroid and everything in its reach. A common tactic is, for example, to freeze an asteroid with a stasis missile, and then send more and more missiles and ships to the asteroid. Then, once the stasis wears off, all your missiles and ships will start striking at once. This game also has a very good multiplayer mode and can keep you busy for hours and hours playing just one map.

Z

Another of my favorite DOS strategy games is called Z. In this game, you can't build new buildings to produce units, but you can conquer sectors that have predefined factories, which can produce different type of units. This game is all about tactics and speed. The more sectors you have, the faster new units are build. The more sectors the enemy has, the faster he builds units, and the harder it becomes for you to fight back, so it's very important to be quick in conquering sectors and holding them.

The game is rather hard, but has a very unique style of humor with funny cut scenes and a very good fighting style. The graphics are really nice and timeless, and there are many interesting units to discover and build. If you are a fan of fast tactics games, Z is definitely for you.

Figure 8 - Z is a very nice game about robots fighting the war of the future about territories. Don't anger General Zod

Honorable mentions

There are many more strategy games available for DOS, such as Wing Commander Armada, which combines strategy elements of harvesting planets, building defenses, and creating new spacecrafts, with the elements of a wing commander shooter in case you encounter an enemy or need to defend your system. Even with a weak fighter, you can

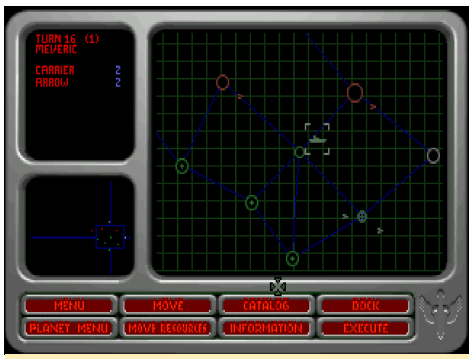


Figure 9 - Wing Commander Armada, build fleets, mine resources, and dog fight your enemies

defend of an entire enemy fleet if you have the right piloting skills.

The classic Civilization is a very well-known series that started on DOS as well, and with which Sid Meier wrote gaming history. Many more games followed the Civilization series, and by now nearly everyone recognizes the name Sid Meier from various games. Besides Civilization, I always loved playing Colonization, which is a game about the colonization of America and the following War of Independence. Colonization is one of my all-time favorites on the Amiga.

There are many other famous DOS strategy games like Constructor, where you own a building company and build houses for people of every class. You can fight man-sized cockroaches openly in the field, or dominate by hiring goons that terrorize the competition. Krush Kill 'N Destroy (KKnD) is a post-apocalyptic strategy game combining elements from Command and Conquer with Fallout and Mad Max elements, creating a very unique setting for a strategy game.

This list of good strategy games on DOS is robust enough to keep you busy for months of play, and thanks to the ODROID platform, you can replay these games on your modern TV. Relive or experience for the first time these awesome gems and pioneers of gaming history!

ODROID-XU4 CASE

A SLEEK, MODERN AND SILENT ENCLOSURE

by @Chris Oi

I made my own machined housing for my ODROID-XU4. It is made of aircraft aluminum with passive cooling using a copper core. I first created a 3D file of the XU4 using Inventor to make it easier to construct the housing, which is available for download at http://bit.ly/1Jfd35b. The images below detail the progress of the build, which consists of a bottom piece raised with rubber feet, and a top piece with cooling fins and a custom ODROID logo.

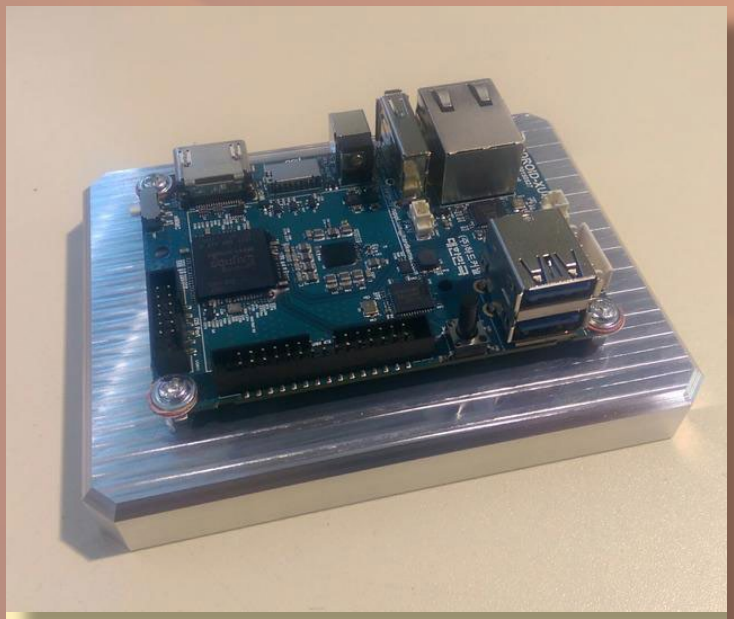


Figure 1 - ODROID-XU4 attached to the base

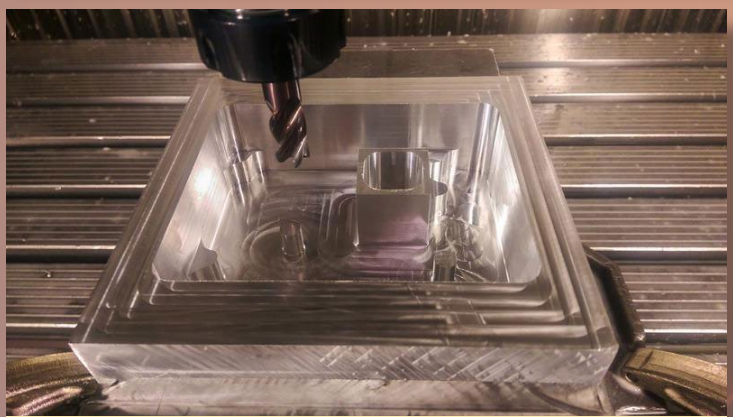


Figure 2 - Milling the top inside part of the case

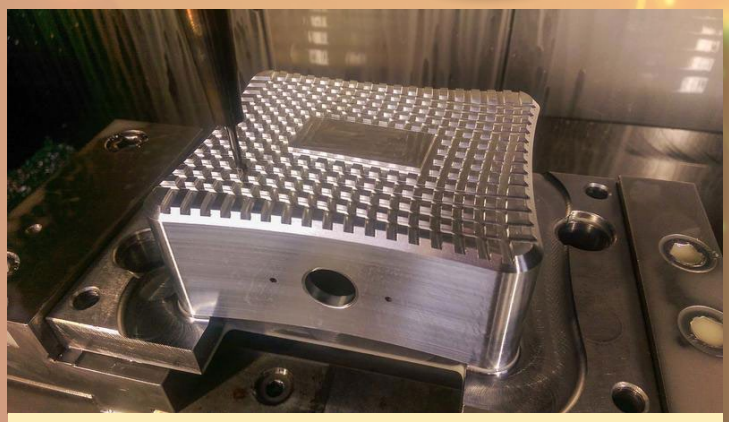


Figure 3 - Creating the cooler fins on the top outside of the case

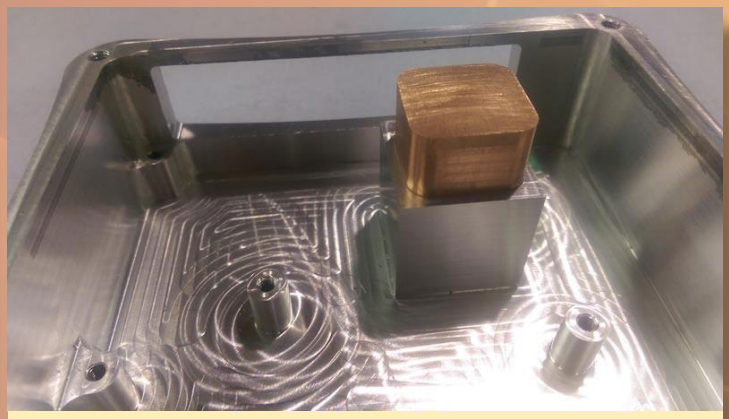


Figure 4 - The copper core attached to the inside of the case for passive cooling

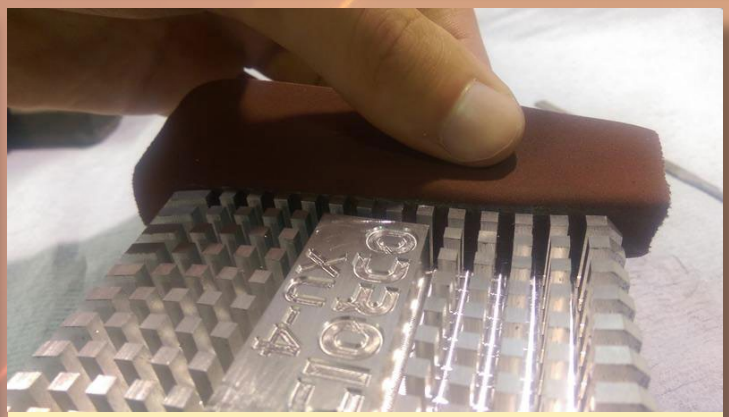


Figure 5 - Polishing the final prototype

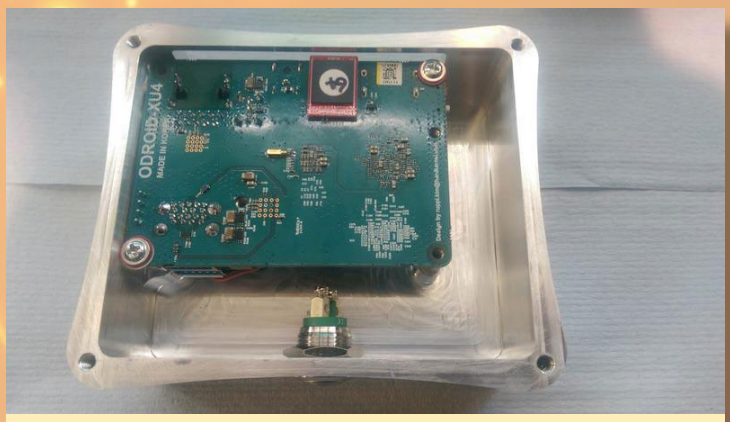


Figure 6 - ODROID-XU4 inside the top cooling area

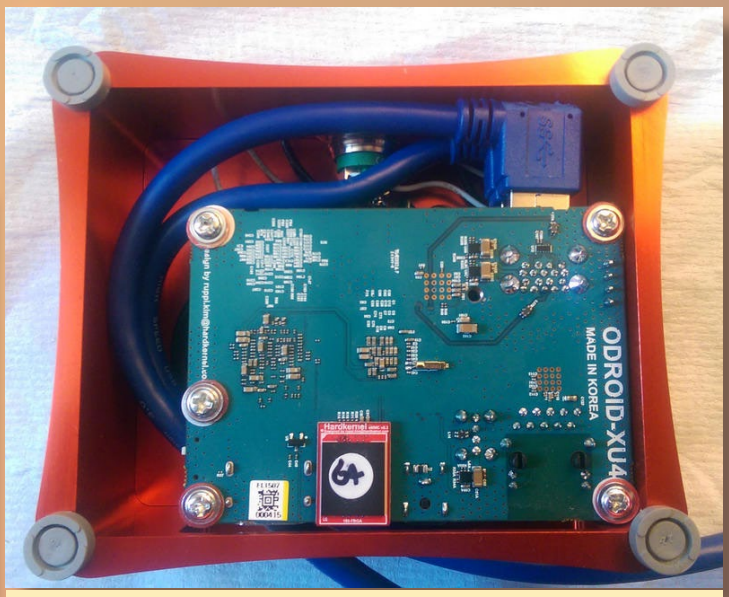


Figure 9 - Final assembly of the case with the bottom removed

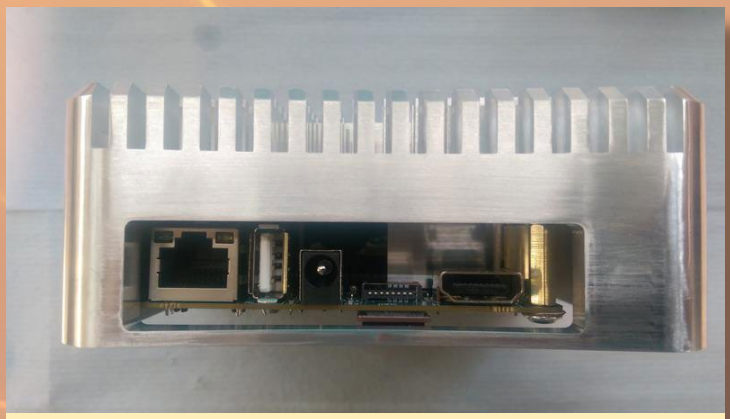


Figure 7 - Back view of the case

Figure 10 - The finished product

Figure 8 - The final product was done with red anodizing

For comments, questions and suggestions, please visit the original post at `http://bit.ly/1LjgnM4`.

This is our attempt at replicating the ODROID-XU4 aluminum case - we think we totally nailed it!

OS SPOTLIGHT

TIZEN FOR ODROID-XU4

edited by Andrew Ruggeri

Tizen is an embedded Linux-based operating system which runs on a wide range of devices including smartphones, tablets, smart TVs, PCs, smart cameras, wearable computing such as smartwatches, Blu-ray players, printers and smart home appliances. Its purpose is to offer a consistent user experience across devices. Tizen is produced by Samsung and Intel, so it's no surprise that its lineage can be traced back to Bada and MeeGo. This article details the steps needed to create a Tizen 3 bootable image or the ODROID-XU3 and XU4. The original article may be found at bit.ly/1O52IhH.

Prepare Binary

To make a bootable card, you will need to get the all the files listed in the following table saved to a single directory on a host PC capable of writing to a microSD card:

Name	Type
bl1.bin.hardkernel	binary
bl2.bin.hardkernel.1mb_uboot	binary
tzsw.bin.hardkernel	binary
u-boot-mmc.bin	binary
sd_fusing_xu4.sh	shell script

To obtain these files, first download the pre-bootloader binaries by typing the following commands into a Terminal window:

```
$ wget https://github.com/hardkernel/u-boot/raw/
odroidxu3-v2012.07/\
sd_fuse/hardkernel_1mb_uboot/bl1.bin.hardkernel
$ wget https://github.com/hardkernel/u-boot/raw/
odroidxu3-v2012.07/\
sd_fuse/hardkernel_1mb_uboot/bl2.bin.hardkernel.1mb_
uboot
$ wget https://github.com/hardkernel/u-boot/raw/
odroidxu3-v2012.07/\
sd_fuse/hardkernel_1mb_uboot/tzsw.bin.hardkernel
```

Download the latest u-boot and kernel from bit.ly/1QGBzDW, extract the contents of the image tarball, and change the u-boot binary names:

```
$ tar xvf tizen-tv_xxxxxxxx.x_\
tv-boot-armv7l-odroidxu3.tar.gz
```

Save the following script into file a named "sd_fusing_xu4.sh":

```
#!/bin/bash

declare FORMAT=""
declare DEVICE=""

# Binaires array for fusing
declare -a FUSING_BINARY_ARRAY
declare -i FUSING_BINARY_NUM=0

declare CONV_ASCII=""
declare -i FUS_ENTRY_NUM=0
declare -r FUSING_IMG="fusing.img"

# binary name | part number | offset | bs
declare -a PART_TABLE=(
      "bl1.bin.hardkernel"          ""      1
512
      "bl2.bin.hardkernel.1mb_uboot" ""      31
512
      "u-boot-mmc.bin"              ""      63
512
      "tzsw.bin.hardkernel"         ""      2111
512
      "params.bin"                  ""      6272
512
      "boot.img"                    1       0
512
      "rootfs.img"                  2       0
4M
      "system-data.img"             3       0
4M
```

```
	"user.img"
5	0	4M
	"modules.img"
6	0	512
	$FUSING_IMG
""	3072	512
	)

declare -r -i PART_TABLE_ROW=4
declare -r -i PART_TABLE_
COL=${#PART_TABLE[*]}/${PART_TA-
BLE_ROW}

# partition table support
function get_index_use_name () {
	local -r binary_name=$1

	for ((idx=0;idx<$PART_TA-
BLE_COL;idx++)); do
		if [ ${PART_
TABLE[idx * ${PART_TABLE_ROW} +
0]} == $binary_name ]; then
			return $idx
		fi
	done

	# return out of bound index
	return $idx
}

# fusing feature
function convert_num_to_ascii ()
{
	local number=$1

	CONV_ASCII=$(printf
\\$(printf '%03o' $number))
}

function print_message () {
	local color=$1
	local message=$2

	tput setaf $color
	tput bold
	echo ""
	echo $message
	tput sgr 0
}

function add_fusing_entry () {
	local name=$1
	local offset=$2
	local size=$3

	FUS_ENTRY_NUM=$((FUS_ENTRY_
NUM + 1))

	echo -n "$name" > entry_
name
	cat entry_name /dev/zero |
head -c 32 >> entry

	echo -n "" > entry_offset
	for ((i=0; i < 4; i++))
	do
		declare -i var;
		var=$(( ($offset >>
(i*8)) & 0xFF ))
		convert_num_to_
ascii $var
		echo -n $CONV_ASCII
> tmp
		cat tmp /dev/zero |
head -c 1 >> entry_offset
	done
	cat entry_offset /dev/zero
| head -c 4 >> entry

	echo -n "" > entry_size
	for ((i=0; i < 4; i++))
	do
		declare -i var;
		var=$(( ($size >>
(i*8)) & 0xFF ))
		convert_num_to_
ascii $var
		echo -n $CONV_ASCII
> tmp
		cat tmp /dev/zero |
head -c 1 >> entry_size
	done
	cat entry_size /dev/zero |
head -c 4 >> entry

	rm tmp
	rm entry_name
	rm entry_offset
	rm entry_size
}

function make_fusing_header () {
	# header magic
	echo -n "BFUS" > fus_hdr_
magic
	cat fus_hdr_magic | head -c
4 > fus_hdr

	# entry number: 1 byte
	convert_num_to_ascii $FUS_
ENTRY_NUM
	echo -n $CONV_ASCII > fus_
hdr_entry_num
	cat fus_hdr_entry_num /dev/
zero | head -c 4 >> fus_hdr

	rm fus_hdr_magic
	rm fus_hdr_entry_num
}

function make_fusing_struct {
	if [ -f entry ];then
		make_fusing_header
		cat fus_hdr entry /
dev/zero | head -c 512 > $FUS-
ING_IMG
		rm fus_hdr entry

		# Write Fusing
Magic Number */
		fusing_image $FUS-
ING_IMG
		rm $FUSING_IMG
	fi
}

function fusing_image () {
	local -r fusing_img=$1

	# get binary info using
basename
	get_index_use_name $(base-
name $fusing_img)
	local -r -i part_idx=$?

	if [ $part_idx -ne $PART_
TABLE_COL ];then
		local -r
device=$DEVICE${PART_
TABLE[${part_idx} * ${PART_TABLE_
ROW} + 1]}
		local -r
seek=${PART_TABLE[${part_idx} *
```

```
${PART_TABLE_ROW} + 2]}
            local -r bs=${PART_
TABLE[${part_idx} * ${PART_TABLE_
ROW} + 3]}
      else
            echo "Not supported
binary: $fusing_img"
            return
      fi

      local -r input_size=`du -b
$fusing_img | awk '{print $1}'`

      print_message 2 "[Fusing
$1]"
      dd if=$fusing_img | pv
-s $input_size | dd of=$device
seek=$seek bs=$bs

      if [ $(basename $fusing_
img) == "u-boot-mmc.bin" ];then
            add_fusing_entry
"u-boot" $seek 2048
      fi
}

function fuse_image_tarball () {
      local -r filepath=$1
      local -r temp_dir="tar_tmp"

      mkdir -p $temp_dir
      tar xvf $filepath -C $temp_
dir
      cd $temp_dir

      for file in *
      do
            fusing_image $file
      done

      cd ..
      rm -rf $temp_dir
      eval sync
}

function fuse_image () {

      if [ "$FUSING_BINARY_NUM"
== 0 ]; then
            return
      fi

      for ((fuse_idx = 0 ; fuse_
idx < $FUSING_BINARY_NUM ; fuse_
idx++))
      do
            local
filename=${FUSING_BINARY_
ARRAY[fuse_idx]}

            case "$filename" in
                *.tar | *.tar.
gz)
                    fuse_image_
tarball $filename
                    ;;
                *)
                    fusing_image
$filename
                    ;;
            esac
      done
      echo ""
}

# partition format
function mkpart_3 () {
      local -r DISK=$DEVICE
      local -r SIZE=`sfdisk -s
$DISK`
      local -r SIZE_MB=$((SIZE >>
10))

      local -r BOOT_SZ=64
      local -r ROOTFS_SZ=3072
      local -r DATA_SZ=512
      local -r MODULE_SZ=20

      let "USER_SZ = $SIZE_MB -
$BOOT_SZ - $ROOTFS_SZ - $DATA_SZ
- $MODULE_SZ - 4"

      local -r BOOT=boot
      local -r ROOTFS=rootfs
      local -r SYSTEMDATA=system-
data
      local -r USER=user
      local -r MODULE=modules

      if [[ $USER_SZ -le 100 ]]
      then
            echo "We recommend
to use more than 4GB disk"
            exit 0
      fi

      echo "====================
==================="
      echo "Label          dev
size"
      echo "====================
==================="
      echo $BOOT"            "
$DISK"1       " $BOOT_SZ "MB"
      echo $ROOTFS"          "
$DISK"2       " $ROOTFS_SZ "MB"
      echo $SYSTEMDATA"      "
$DISK"3       " $DATA_SZ "MB"
      echo "[Extend]""       "
$DISK"4"
      echo " "$USER"         "
$DISK"5       " $USER_SZ "MB"
      echo " "$MODULE"
" $DISK"6     " $MODULE_SZ "MB"

      local MOUNT_LIST=`mount |
grep $DISK | awk '{print $1}'`
      for mnt in $MOUNT_LIST
      do
            umount $mnt
      done

      echo "Remove partition
table..."
      dd if=/dev/zero of=$DISK
bs=512 count=16 conv=notrunc

      sfdisk --in-order --Linux
--unit M $DISK <<-__EOF__
      4,$BOOT_SZ,0xE,*
      ,$ROOTFS_SZ,,-
      ,$DATA_SZ,,-
      ,,E,-
      ,$USER_SZ,,-
      ,$MODULE_SZ,,-
      __EOF__

      mkfs.vfat -F 16 ${DISK}1 -n
$BOOT
      mkfs.ext4 -q ${DISK}2 -L
$ROOTFS -F
      mkfs.ext4 -q ${DISK}3 -L
$SYSTEMDATA -F
```

```
	mkfs.ext4 -q ${DISK}5 -L
$USER -F
	mkfs.ext4 -q ${DISK}6 -L
$MODULE -F
}

function show_usage () {
	echo "- Usage:"
	echo "  sudo ./sd_fusing*.
sh -d <device> [-b <path> <path>
..] [--format]"
}

function check_partition_format
() {
	if [ "$FORMAT" != "2" ];
then
		echo "-------------
----------"
		echo "Skip $DEVICE
format"
		echo "-------------
----------"
		return 0
	fi

	echo "---------------------
----------"
	echo "Start $DEVICE format"
	echo ""
	mkpart_3
	echo "End $DEVICE format"
	echo "---------------------
----------"
	echo ""
}

function check_args () {
	if [ "$DEVICE" == "" ];
then
		echo "$(tput setaf
1)$(tput bold)- Device node is
empty!"
		show_usage
		tput sgr 0
		exit 0
	fi

	if [ "$DEVICE" != "" ];
then
		echo "Device: $DE-
VICE"
	fi

	if [ "$FUSING_BINARY_NUM"
!= 0 ]; then
		echo "Fusing bina-
ry: "
		for ((bid = 0 ; bid
< $FUSING_BINARY_NUM ; bid++))
		do
			echo "
${FUSING_BINARY_ARRAY[bid]}"
		done
		echo ""
	fi

	if [ "$FORMAT" == "1" ];
then
		echo ""
		echo "$(tput setaf
3)$(tput bold)$DEVICE will be
formatted, Is it OK? [y/n]"
		tput sgr 0
		read input
		if [ "$input" ==
"y" ] || [ "$input" == "Y" ];
then
			FORMAT=2
		else
			FORMAT=0
		fi
	fi
}

function print_logo () {
	echo ""
	echo "Odroid-XU4 download-
er, version 0.5"
	echo "Authors: Inha Song
<ideal.song@samsung.com>"
	echo ""
}

print_logo

function add_fusing_binary() {
	local declare binary_
name=$1
	FUSING_BINA-
RY_ARRAY[$FUSING_BINARY_
NUM]=$binary_name
	FUSING_BINARY_
NUM=$((FUSING_BINARY_NUM + 1))
}


declare -i binary_option=0

while test $# -ne 0; do
	option=$1
	shift

	case $option in
	--f | --format)
		FORMAT="1"
		binary_option=0
		;;
	-d)
		DEVICE=$1
		binary_option=0
		shift
		;;
	-b)
		add_fusing_binary
$1
		binary_option=1
		shift
		;;
	*)
		if [ $binary_option
== 1 ];then
			add_fusing_
binary $option
		else
			echo "Unkown
command: $option"
			exit
		fi
		;;
	esac
done

check_args
check_partition_format
fuse_image

make_fusing_struct
```

Make the file executable by opening a Terminal window in the directory where the bash script was created and typing

the following:

```
$ chmod u+x sd_fusing_xu4.sh
```

Next, install the pv tools:

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

Installing to microSD card

Connect the microSD card using a card reader and check the device node.

```
$ sudo fdisk -l
```

The following is an example output from the fdisk command.

```
..........
Partition table entries are not
in disk order
Disk /dev/sdb: 32.0 GB,
32010928128 bytes
64 heads, 32 sectors/track, 30528
cylinders, total 62521344 sectors
Units = sectors of 1 * 512 = 512
bytes
Sector size (logical/physical):
512 bytes / 512 bytes
I/O size (minimum/optimal): 512
bytes / 512 bytes
Disk identifier: 0x00000000
Device Boot Start End Blocks Id
System
/dev/sdb1 * 8192 139263 65536 e
W95 FAT16 (LBA)
..........
```

Run "sd_fusing_xu4.sh" followed by the device node, which is /dev/sdb in the following example. Note that it can take a few minutes for the script to finish.

```
$ sudo ./sd_fusing_xu4.sh -d \
/dev/sdb -b bl1.bin.hardkernel \
bl2.bin.hardkernel.1mb_uboot \
tzsw.bin.hardkernel \
u-boot-mmc.bin
```

Insert your microSD card into the ODROID and select SD as the boot mode on the hardware switch.

Installing to eMMC module

Connect the eMMC module to a desktop computer using a card reader and check the device node:

```
$ sudo fdisk -l
..........
Partition table entries are not
in disk order
Disk /dev/sdb: 7948 MB,
7948206080 bytes
245 heads, 62 sectors/track, 1021
cylinders, total 15523840 sectors
Units = sectors of 1 * 512 = 512
bytes
Sector size (logical/physical):
512 bytes / 512 bytes
I/O size (minimum/optimal): 512
bytes / 512 bytes
Disk identifier: 0x00000000
Device Boot Start End Blocks Id
System
/dev/sdb1 * 8192 139263 65536 e
W95 FAT16 (LBA).......
```

Run "sd_fusing_xu4.sh" script followed by the device node

```
$ sudo ./sd_fusing_xu4.sh -d /
dev/sdb --format
```

Press the "y" key to format, which can take a few minutes. Wait for the script to finish. The following is example output from the script:

```
Device: /dev/sdb
/dev/sdb will be formatted, Is it
OK? [y/n]
y
------------------------------
Start /dev/sdb format
..........
```

Next, download the latest boot and platform image from bit.ly/1J2ytri and bit.ly/1S6BEjE. Connect the eMMC to the desktop PC using a card reader and check the device node:

```
$ sudo fdisk -l
..........
Partition table entries are not
in disk order
Disk /dev/sdb: 7948 MB,
7948206080 bytes
245 heads, 62 sectors/track, 1021
cylinders, total 15523840 sectors
Units = sectors of 1 * 512 = 512
bytes
Sector size (logical/physical):
512 bytes / 512 bytes
I/O size (minimum/optimal): 512
bytes / 512 bytes
Disk identifier: 0x00000000
Device Boot Start End Blocks Id
System
/dev/sdb1 * 8192 139263 65536 e
W95 FAT16 (LBA).......
```

Run "sd_fusing_xu4.sh" script followed by the device node. Note that "/dev/sdb" is used as in the example commands below:

```
$ sudo ./sd_fusing_xu4.sh -d /
dev/sdb -b \
tizen-tv_20150824.1_tv-boot-arm-
v7l-odroidxu3.tar.gz \
tizen-tv_20150824.1_tv-wayland-
armv7l-odroidu3.tar.gz
```

Boot into Tizen

Insert your microSD card or eMMC module into the ODROID and select the SD boot mode via the hardware switch. Turn on the power, enter the u-boot prompt, and run the following commands:

```
# mmc dev 0
# mmc read 0x50000000 0x1 0xa3e
# mmc dev 1 1
# mmc write 0x50000000 0x0 0xa3e
# mmc dev 1 0
```

Next, change the boot mode to eMMC, then reboot. Now, you can use the eMMC for Tizen development.

MEET AN ODROIDIAN

GEORG MILL, INNOVATIVE AND CREATIVE HARDWARE MAKER

edited by Rob Roy

Please tell us a little about yourself.

My name is Georg Mill. My wife and I live in Düsseldorf Germany. As the Düsseldorf website says, "Düsseldorf is the bustling Rhine metropolis in the heart of Europe with much to offer residents and visitors." I've enjoyed living in this city for about 10 years now.

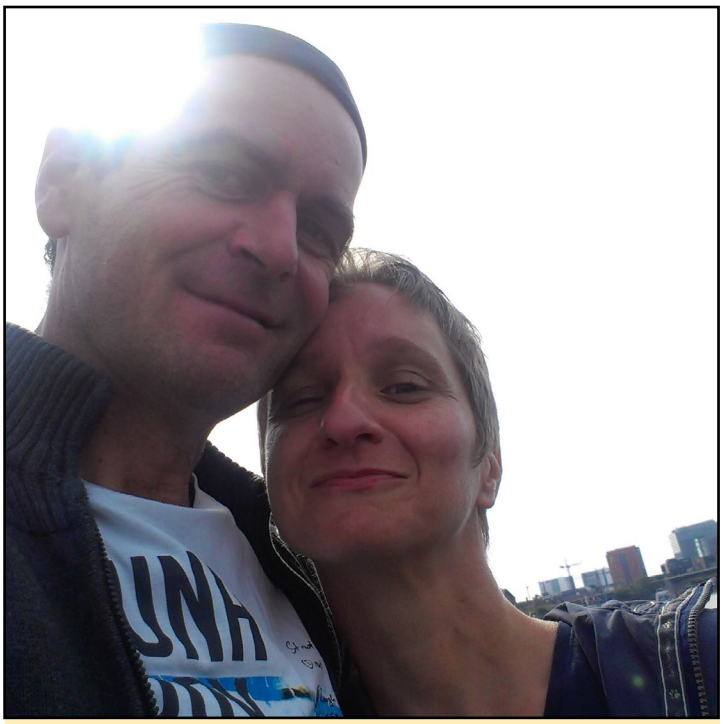
How did you get started with computers?

Computer scientist is my second job. My first job was as an offset printer. So I do not come from a University, but learned the basics in Heidelberg in 1999 at a technical school for computer science with a focus on web development.

My very first contact with computers came from my brother, who built a DIY Pong hardware player when he was 17 years old in the early 1970s. This device could be connected to a TV. My parents gave us a gift on Christmas of that year: an electronic hardware kit to build things like an FM radio yourself. In those days, computers were rare and rather expensive, so we didn't have one. Even in our school, nothing like that was available. I finally got my first computer when I finished my secondary school exams. Smartphones and handhelds weren't invented at that time. Nobody new what a "maker" is, but we played with the Elektronikus experimenter called "Kosmos Elektronikus Radio+Elektronik 1 and 2".

With these electronic parts, my brother and I were able to build our own, fully working FM radio. My parents got big eyes when we proudly presented that little wired thing to them.

Kosmos Elektronikus Radio



Georg and his wife

That happened a long time ago. I am 51 years old now, and electronic devices are cheap, easy to use, and available to almost everyone now.

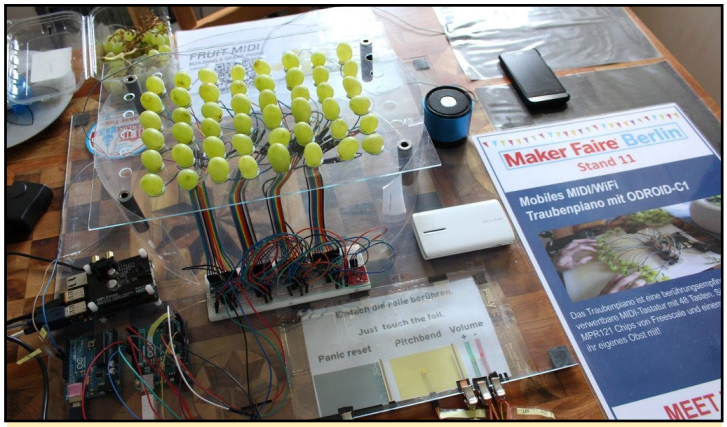
What attracted you to the ODROID platform?

Some years ago, I started to play around with an Arduino, and that reminded me of the early days of my childhood. We would have been happy if devices like ODROIDs were available then. I used the Arduino to build a DIY electronic drum set from scratch. The drum pads were self-made meshed drum triggers made of fly screen and wooden rings and some piezos. I eventually made a better version, which is described at `http://bit.ly/1Yduav0`. I called it YAAMI-Drum (`http://bit.ly/1Qfa18W`) and it did what I wanted it to do: a silent way of playing drums in our house without disturbing the neighbors living right beneath us.

Everybody knows the Raspberry Pi, which was my first Single Board Computer (SBC). I found out that these little things are even better than a "normal" computer because you can connect a lot of sensors, inputs and outputs to their GPIOs. I was happy to develop my first own little camera surveillance and networking webradio with it. Then, I tried to build a solution for realtime audio processing live music and found the original ODROID-C1. The Odroid-C1+ seems to be used a

lot for games and for Kodi (formerly known as XBMC), but I used it to develop my own real-time kernel (rt_preempt) in order to play live music with it. Another project is a timelapse rag for a real EOS DSLR camera. Some nice examples can be found at http://bit.ly/1ErAzMK, which was a first inspiration for me. My version can be found at http://bit.ly/1Ydufi7 and http://bit.ly/1miU5pb.

One of my favorite projects with the Odroid-C1+ is an audio project called "Traubenpiano" which means "grape piano". It is a full-featured WiFi-enabled MIDI keyboard (http://bit.ly/1UgIRgE) that has some very special devices as keys: grapes! It was first demonstrated at the Maker Faire Berlin 2015 http://bit.ly/1NjEf6L and Codemotionworld 2015 in Berlin (http://bit.ly/1jX8fKV). You will see it next time here in Germany at the Makerfaire-ruhr 2016 in March (http://bit.ly/1OqeaAA).



Berlin 2015

Which ODROID is your favorite?

Since I only own an ODROID-C1+ at the time of this writing, the answer is simple: the ODROID-C1+ with an audio shield on top. However, in the future, I'll try to get an XU4 which is also a nice platform for a grape piano!

Your hardware projects, especially the graped piano and disco banana, are very creative. What motivated you to build them?

Of course, the "grape piano" is nothing that you can use for serious music production. It is a project that I developed to support the children in a hospice for little children here in Düsseldorf called "Rainbowland," or as we say in German "Regenbogenland" (http://bit.ly/1Oqetvh). The children suffer from some very rare diseases, so rare that there is no hope for them to get some medicine that would help them to get back into a so-called normal, healthy life. Most of these children like to listen to music, or even try to make music themselves. Because most of them have lost the control of their muscles, they have to find another way of playing. For that reason, I started to develop some special touch sensitive triggers that allowed them to make music without needing to move their fingers or arms too much. The device has low power consumption and is easy to use, which is how these instruments should work. At the moment, I am still working on the development, and it is far from being ready to use.

What innovations would you like to see in the future Hardkernel products?

It would be nice if the ODROID-XU4 could get a Hi-Fi shield like the one available for the ODROID-C1+, or at least good quality audio input and output.

What hobbies and interest do you have apart from computers?

At the age of 15, my father bought me a really simple Jazz drum set from a company called Hoshino. As a result, our relationship with our neighbors became a little more difficult. They wanted to watch a soccer match on TV, while I wanted to learn drumming at loud volume. Recently, I learned a lot from some professional jazz drummers here in Düsseldorf and Cologne that it is possible to play drums without disturbing people even sitting in the same room.

I like to travel and go on short journeys with my wife. She is afraid of flying in an airplane, so our travel destinations are limited to what can be reached by car or train. We can often be found on the coast of some little islands of the Netherlands and Denmark during holidays. There is a lot of steady wind, which is the reason why I like to fly stunt kites. Most people believe that I am not able to control them, but the truth is that I love to loop the kites quickly with full control over them.

When programming becomes too heavy, I like to take my mountain bike and ride through the nearby forests nearby and throughout Düsseldorf, which has a lot of trees. If that is not enough, I get some positive energy from practicing Aikido.

What advice do you have for someone want to learn more about programming?

My only advice is to be curious about everything. If you are interested in programming, be aware of the fact that no one was ever born with knowledge about electronic circuits or a scientific IT background. It's a lot of work to learn how to get things running. Big names like Linus Torvalds and Thomas Gleixner started their careers somewhere. Today, you do not even need to be an engineer or computer scientist to get some basic home automation running. You only need to be curious about how things work. Don't think too big. Starting with little goals and smal projects often makes more fun than developing a super-computer with thousands of ODROIDS, even if it is only a grape piano!