Linux/Android Gaming • Gentoo for ODROID-C2 • Compiling Android

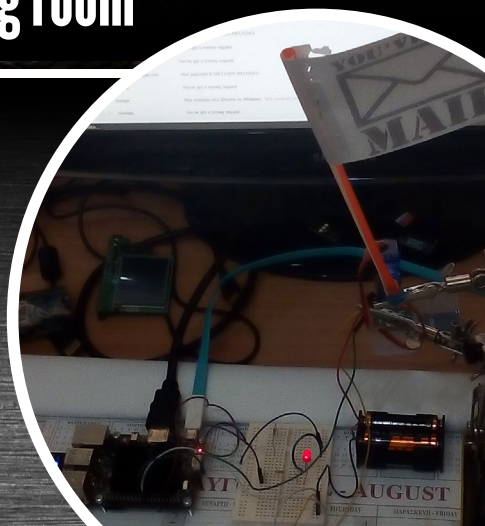
ODROID Magazine

Year Three
Issue #34
Oct 2016

High end gaming using ODROID-C2 and GameStream

Bring your favorite modern game titles from your office to your living room

- Remotely Monitor Modbus Registers using an ODROID-XU4
- An ODROID IoT Device and Application: Gmail Mechanical Notifier



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

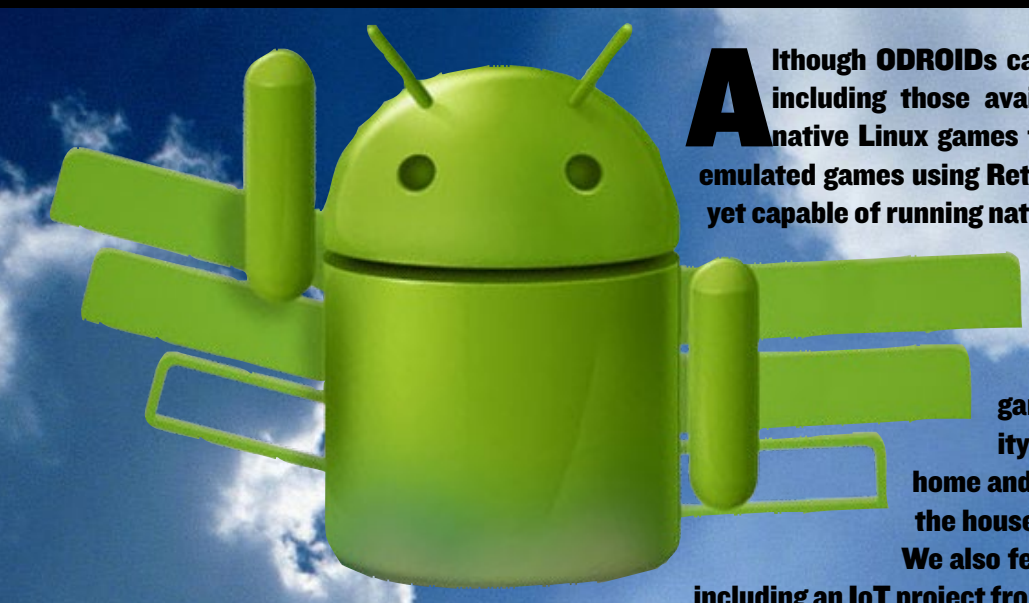
We are now shipping the ODROID-U3 device to EU countries! Come and visit our online store to shop!

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



Although ODROIDs can already play thousands of games, including those available from the Google Play Store, native Linux games that were ported by developers, and emulated games using RetroArch, many modern games are not yet capable of running natively on the ARM platform. However, NVIDIA publishes a software package that allows streaming of high-end games across a network, allowing ODROIDs to work as a remote gaming station. This gives the flexibility of running a central gaming server at home and playing high-end games anywhere in the house on a large monitor.

We also feature a few DIY projects this month, including an IoT project from Miltiadis that raises a flag when an email arrives. Adrian shows us how to set up an IP webcam, Joel gives an overview of his industrial automation setup, and Bo documents his quest in finding the best cooling setup for an ODROID-XU4. If you want to have complete control over your operating system, we have instructions for compiling both Gentoo and Android Lollipop. We also have some game reviews for our favorite Android and Linux games, including some ports that were recently updated by Tobias for the ODROID-C2.

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/1ypImXs.
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

OUR AMAZING ODROIDIAN STAFF:

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 prebuilt 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`.

Bruno Doiche, Senior Art Editor

Is the GameStream the holy grail that Bruno always wished to play his favorite PC games on his living room when his wife is not around to scoot him back to his dungeon/design lab? Maybe. Just don't tell her that he is considering purchasing yet another computer just to play games. After all, he is already struggling to make all his electronics to fit his apartment, so he avoids the endless call to reunite himself to the PC MASTER RACE!

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.

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.

Andrew Ruggeri, Assistant Editor

I am a Biomedical Systems engineer located in New England currently working in the Aerospace industry. An 8-bit 68HC11 microcontroller and assembly code are what got me interested in embedded systems. Nowadays, most projects I do are in C and C++, or high-level languages such as C# and Java. For many projects, I use ODROID boards, but I still try to use 8bit controllers whenever I can (I'm an ATMEL fan). Apart from electronics, I'm an analog analogue photography and film development geek who enjoys trying to speak foreign languages.

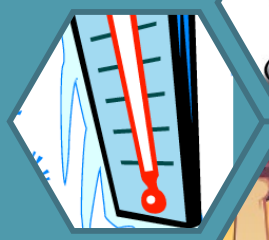
Venkat Bommakanti, Assistant Editor

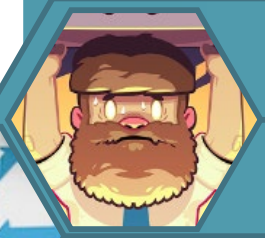
I'm a computer enthusiast from the San Francisco Bay Area in California. I try to incorporate many of my interests into single board computer projects, such as hardware tinkering, metal and woodworking, reusing salvaged materials, software development, and creating audiophile music recordings. I enjoy learning something new all the time, and try to share my joy and enthusiasm with the community.

Josh Sherman, Assistant Editor

I'm from the New York area, and volunteer my time as a writer and editor for ODROID Magazine. I tinker with computers of all shapes and sizes: tearing apart tablets, turning Raspberry Pis into PlayStations, and experimenting with ODROIDs and other SoCs. I love getting into the nitty gritty in order to learn more, and enjoy teaching others by writing stories and guides about Linux, ARM, and other fun experimental projects.

INDEX





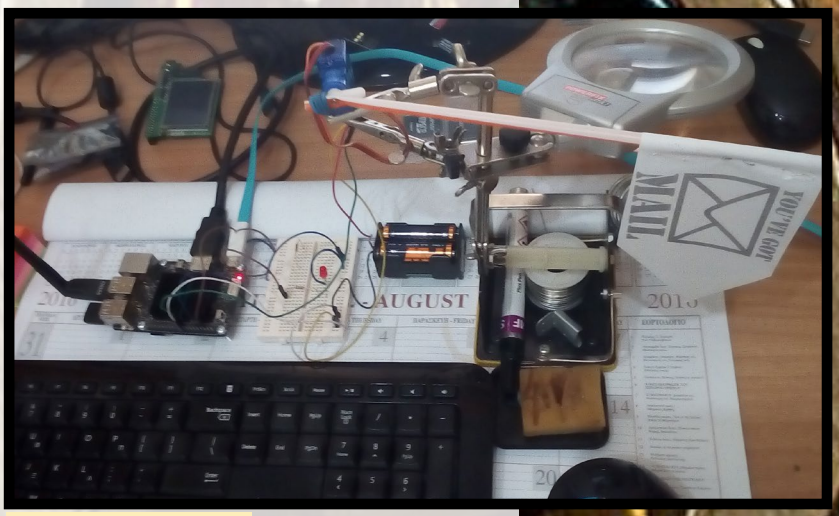
GMAIL MECHANICAL NOTIFIER

AN ODROID IOT DEVICE AND APPLICATION

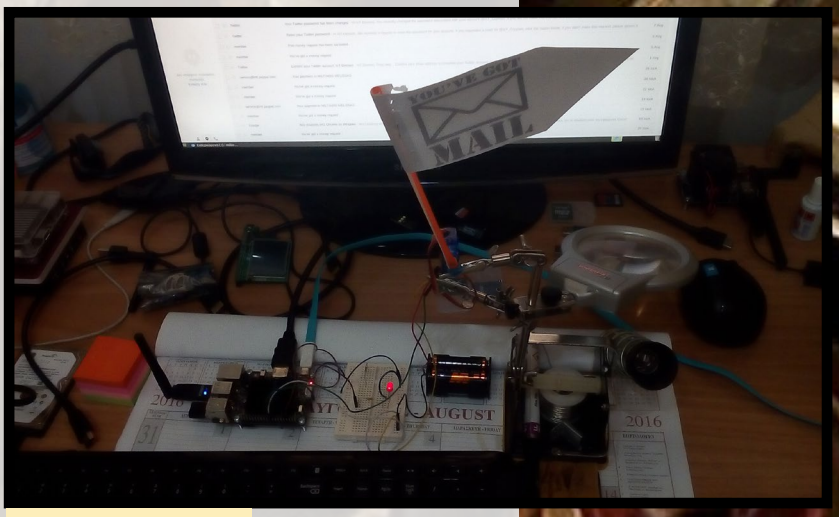
by Miltiadis Melissas

Following my last article in the September issue of ODROID Magazine titled "ODROID-C2 as an IoT device: Interfacing with the real world", I was looking for an Internet of Things (IoT) application that made use of a servo motor. This tutorial details my servo motor project, through an enjoyable process of constructing an IoT device which is constantly checking your Gmail account for any incoming messages.

The IoT device employing an ODROID-C2 under the hood, logins automatically to your gmail account, and checks for new incoming messages. If such messages exist, an LED lights up and a mechanical mail-flag is raised to notify you with the banner that declares "YOU'VE GOT MAIL!" When all of the new messages have been read by the user, the mail flag moves back down and the LED goes off. Watch the video on my Youtube channel at `http://bit.ly/2bT9bMz` to see the device in action.



Flag down - LED off



Flag up - LED on

System requirements

You will need an ODROID-C2 with Hardkernel's latest Ubuntu v2.0 release (`http://bit.ly/2cBibbk`) and Python Ver. 2.7.12 installed. Additional materials include:

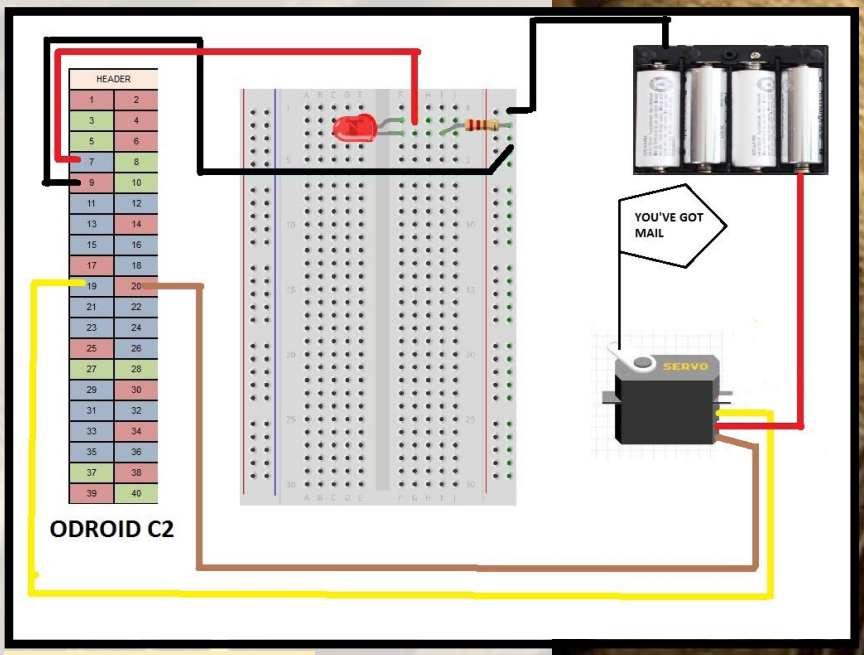
- 1 x ODROID-C2 (ARM 64bit 1.5Ghz quad core single board computer)
- 1 x Gmail account (which is free)
- 1 x Breadboard
- 1 x LED (Light Emitted Diode)
- 1 x 220 ohm Resistor
- 1 x Servo motor (link)
- 4 x 1.5 Batteries and of course (6V)
- 1 x set of Jumpers

Hardware connections

Please refer to the schematic in Figure 3 and to the Hardkernel's excellent pin-layout diagram for the ODROID-C2 at `http://bit.ly/2aXAlmt`. The LED's anode(+) is connected to pin7 through a

resistor (220Ω) while its cathode(-) to the ground (pin9). The two pins are lying out, side by side according to the Hardkernel's pin layout.

Now let's explore how to use a servo to control the mail flag. The servo is attached to pins 19 and 20. However, since it requires more current than the ODROID-C2 can supply, we will need a set of the four (4) batteries as an added power supply. The servo has three (3) wires: yellow, red and brown. The yellow carries the signal, which is the pulse width modulation (PWM) and its attached to pin19. Please refer again to Hardkernel documentation page at http://bit.ly/2ckfdKn for the pins that can provide those kind of pulses. For the ground(-), the servo is connected to pin20, using the brown wire. However, it's better to use a common ground and leave it unplugged as pins are precious commodity, especially for more complicated projects. Finally, the red wire of servo is connected to the power bank, in other words, with the four batteries (+Vcc). So it is red to red, leaving the blank wire of the power bank for the last, for the connection with the common ground as its negative(-). The hardware phase of the Gmail Mechanical Notifier is now ready. Let us spring it to life with the Python script we will write for this purpose!



Gmail notifier schematic

Preliminary software

Before we start writing the Python script, let us examine the ODROID-C2 operating systems (OS). All boards made by Hardkernel can run either Linux or Android, and the ODROID-C2 is no different. For the Gmail Mechanical Notifier, we will use Linux as the main OS. The reason is that Linux is more versatile and robust when it comes to the Internet of Things (IoT) applications. You can flash Linux Ubuntu 16.04 Mate following the guide at `http://bit.ly/1Vk9u4o`.

Finally, install the WiringPi2 library. This library controls the pins on ODROID-C2. Hardkernel provides an excellent guide on their site for installing the WiringPi2 library at `http://bit.ly/2ba6h8o`.

You will need additional utilities if you manually rebuild the bindings with swig-python WiringPi. They can be installed using the following command:

```
$ sudo apt-get install python-dev python-setuptools
```

Download and setup WiringPi2 for Python from the repository:

```
$ git clone https://github.com/hardkernel/WiringPi2-Python.git
$ cd WiringPi2-Python
$ git submodule init
$ git submodule update
```

Build and install the library:

```
$ sudo python setup.py install
```

Download and run the example source code from http://bit.ly/2cKcwkd, which is optional and only for testing purposes.

```
$ wgeg http://dn.odroid.com/source_peripherals/ctinkeringkit/example-led.py
$ sudo python example-led.py
```

You can also use a python IDE called IDLE by running the following command:

```
$ sudo apt-get install idle
```

Application software

Run the IDLE utility and copy and paste the lines of code indicated below. When done editing, save the Python script under the name of trace_messages.py in the directory /home/odroid/Documents/gmail_Python. If you don't have IDLE installed, you can copy and paste the script to any installed editor on your system and save it under the same name.

I will discuss each line of code to explain what is going on inside this script:

```
#! python
import gmail, wiringpi2 as wpi, time # import the modules

#use ODROID-C2 pin numbers for LED and SERVO
LED_PIN=7
SERVO_PIN=12

wpi.wiringPiSetup()

#setup pin (LED) as an output
wpi.pinMode(LED_PIN,1)

#setup pin (SERVO) as an output too
wpi.pinMode(SERVO_PIN,1)


#setup Pulse Width Modulation(PWM) for Servo
wpi.softPwmCreate(SERVO_PIN,0,50)

#setup
g = gmail.login('youremail@gmail.com', 'yourpassword')
unread_messages = g.inbox().mail(unread=True)
total_messages = 0

for message in unread_messages:
      total_messages += 1

if total_messages > 0:
      # there are unread emails, turn light on
      wpi.digitalWrite(LED_PIN,1)
      for i in range (15):
         wpi.softPwmWrite(SERVO_PIN,i)
         time.sleep(0.2)
else:
      # there are no unread emails, turn light off
      wpi.digitalWrite(LED_PIN,0)
      for i in range (15,0,-1):
         wpi.softPwmWrite(SERVO_PIN,i)
         time.sleep(0.2)
```

This script is heavily modified, adapted and upgraded properly for the needs of this project, using http://bit.ly/2cGSwBS as a reference. The basic idea remains the same, however. Let us break up the code and see what is going on:

```
<import gmail, wiringpi2 as wpi, time> # import the modules
```

First, we import the modules. They are three (3) of them: gmail, wiringpi and time. Modules in Python are small pieces of code written for a purpose, similar to libraries in the Arduino IDE. "Wiringpi2" is the module for controlling the pins on ODROID-C2, and that is why we installed this module previously. "Time" is a module built into Python (system module) to provide timing functions. The "gmail" module is written by Charlie Guo (http://bit.ly/2bY7Vhh), and it is very important for our project to work. In order to import it, you have to install it first, which is not that hard. Download the library from Github (http://bit.ly/2cC01Jb) into a known directory and extract the contents there. Inside, there should be a folder called "gmail". Copy this whole folder into the directory /home/odroid/Documents/gmail_python. The module is, in essence, a script that logs into your Gmail account with your credentials and reads your incoming messages.

The next line sets up the wiring to refer to the WiringPi GPIO#, which is the first column according to the Hardkernel's GPIO pin map in Figure 4.

```
<wpi.wiringPiSetup()>
```

These two lines of code are very simple: we define the pins we are going to use.

```
<LED_PIN=7>
<SERVO_PIN=12>
```

Note that we are referring to pin19 providing the pulse width modulation to the servo as pin12. Again, this is according to the pin layout from Hardkernel: pin19 is referred to as WiringPi GPIO#12 according Hardkernel's table already given above (http://bit.ly/2aXAlmt). Fortunately, pin7 remains as pin7 itself.

```
<wpi.pinMode(LED_PIN,1)>
```

We set pin7 driving the LED as an output here:

```
<wpi.pinMode(SERVO_PIN,1)>
```

We are doing the same with the servo too by declaring it as an output:

```
<wpi.softPwmCreate(SERVO_PIN,0,50)>
```

GPIO PIN MAP

ODROID-C2 40pin Layout

Power Pin
Special Function
GPIO/Special Function

WiringPi GPIO#	Export GPIO#	ODROID-C2 PIN	Label	HEADER		Label	ODROID-C2 PIN	Export GPIO#	WiringPi GPIO#
			3V3	1	2	5V0			
	205	I2CA_SDA	SDA1	3	4	5V0			
	206	I2CA_SCL	SCL1	5	6	GND			
7	249	GPIOX.BIT21	#249	7	8	TXD1	TXD_B	113	
			GND	9	10	RXD1	RXD_B	114	
0	247	GPIOX.BIT19	#247	11	12	#238	GPIOY.BIT10	238	1
2	239	GPIOX.BIT11	#239	13	14	GND			
3	237	GPIOX.BIT9	#237	15	16	#236	GPIOX.BIT8	236	4
			3V3	17	18	#233	GPIOX.BIT5	233	5
12	235	GPIOX.BIT7	#235	19	20	GND			
13	232	GPIOX.BIT4	#232	21	22	#231	GPIOX.BIT3	231	6
14	230	GPIOX.BIT2	#230	23	24	#229	GPIOX.BIT1	229	10
			GND	25	26	#225	GPIOY.BIT14	225	11
	207	I2CB_SDA	SDA2	27	28	SCL2	I2CB_SCL	77	
21	228	GPIOX.BIT0	#228	29	30	GND			
22	219	GPIOY.BIT8	#219	31	32	#224	GPIOY.BIT13	224	26
23	234	GPIOX.BIT6	#234	33	34	GND			
24	214	GPIOY.BIT3	#214	35	36	#218	GPIOY.BIT7	218	27
		ADC.AIN1	AIN1	37	38	1V8	1V8		
			GND	39	40	AIN0	ADC.AIN0		

ODROID-C2 GPIO pin map

This is a very important function within the object of wpi inside the script, which sets up the pulse width modulation for the servo. The arguments are the SERVO_PIN (i.e pin12), the initial value ('0'), and the pulse width modulation range ('50'). To make things simple, we keep the pulse high for 5ms made of 50 steps. Of course, you can experiment with other values when you are calibrating your servo. The code represents my values after some experimentation with calibrating the servo with the correct position of the flag.

```
<g = gmail.login('youremail@gmail.com', 'yourpassword')>
```

We create the object "g" and call the login method based on the gmail module imported previously. Substitute your email and password with your real gmail account and password, leaving the quotes in place.

```
<unread_messages = g.inbox().mail(unread=True)>
```

We retrieve all unread messages and store them under the variable "unread_messages". Notice how "unread=True" is passed as a parameter. You can change this to retrieve messages based on different parameters, such as sender or subject.

```
<total_messages = 0>


<for message in unread_messages:
      total_messages += 1>
```

We iterate through unread messages and we increment the variable "total_messages" by one if necessary:

```
<if total_messages > 0:
      # there are unread emails, turn light on
      wpi.digitalWrite(LED_PIN,1)
      for i in range (15):
         wpi.softPwmWrite(SERVO_PIN,i)
         time.sleep(0.2)
else:
      # there are no unread emails, turn light off
      wpi.digitalWrite(LED_PIN,0)
      for i in range (15,0,-1):
         wpi.softPwmWrite(SERVO_PIN,i)
         time.sleep(0.2)>
```

This is a very simple conditional check. If the number of the variable "total_messages" is greater than zero, then we do two things. First, we turn the LED on with the following statement:

```
<wpi.digitalWrite(LED_PIN,1)>
```

Second, we start the servo, which raises the flag by changing the duty cycle:

```
<wpi.softPwmWrite(SERVO_PIN,i)>
```

Otherwise, if there are no unread messages, we turn the LED off and bring the flag down. The flag gets its horizontal position by changing the duty cycle of pulse as we count now counterclockwise within the same range by deducting -1 in each iteration. Please note the following loop:

```
<for i in range {15,0,-1):>
```

Run the script

Now it is time to run our script. Open a terminal (from the GUI go to Applications->System Tools->Mate Terminal) and type the following:

```
$ sudo python /home/odroid/Documents/\
gmail_python/trace_messages.py
```

Then watch what happens. If there are any incoming messages, the flag should be raised and our LED should be set to on. If so, we are successful and our script is working! If not, trace for any mistakes through your code. Next, we need to take a

further step to make it run automatically for given intervals of time, such as every 5 minutes. For this task, we will use the cron utility. What is cron? It defines actually jobs that are used to schedule tasks and scripts, such as deftags, backups and alarms. For more information about cron, please refer to http://bit.ly/2bTmNaN. In order to activate cron, we must execute the command crontab which gives us a list of scheduled tasks:

```
$ crontab -e <Enter>
```

It will probably be empty. Then, choose any text editor and add the following line of code at the end of the scheduled tasks list:

```
*/5 * * * * sudo python /home/odroid/Documents/gmail_python/
\trace_incomings.py
```

The five "stars" ("* * * * *") specify how often you want the task to be run. The first star controls the minutes, that is why I put this '/5' after it, since I want this scheduled task to run every five minutes. The second star controls the hours, the third specifies the day of month, the fourth indicates the month and the fifth represents the day of the week. Those four were intentionally left blank without any '/numbers' besides the stars. You can experiment with other options as well. At the end of the scheduled task, there is the command itself we want to be run automatically:

```
$ sudo python /home/odroid/Documents/gmail_python/trace_incomings.py
```

This command runs our script and points to the path where it is located, which is, in this case, /home/odroid/Documents/gmail_python.

Then, save and close the editor. Now, wait and watch as the application does its magic. Send any message to your gmail account for a test if you do not have any unread ones and see your flag go up. The flag with "You've Got Mail!" should raise at a time perfectly matching with your LED lighting up. Congratulations, your Gmail Mechanical Notifier is now working!

Final notes

You should keep in mind that any Python code in IDLE must be executed as root user, otherwise your code will not work. A simple approach it is just to create a shortcut of IDLE on your desktop after installation and then edit this shortcut with:

```
$ cd ~/Desktop
$ sudo nano idle.desktop
```

and then amend the line "Exec=/usr/bin/idle" to read "Exec=/usr/bin/gksu -u root idle", then save the file.

I hope you enjoyed this project as much as I did when I was developing and writing it. The Gmail Mechanical Notifier is the second one of a series of three projects that I wrote about for ODROID Magazine. My next IoT project uses an ODROID-C2 to observe and control the fermentation of wine bottles in a cellar. In particular, the ODROID-C2 observes and controls the air-conditioning setup by measuring the temperature and humidity of the fermentation environment in the cellar. It notifies the user of any diversions from its acceptable values through some actuators. Any anomalies will update the user's Twitter stream, giving the opportunity for further product analysis. As I always say, "with ODROIDs, the sky's the limit!"

TRANSFORMING YOUR ODROID INTO AN IP WEBCAM

by Adrian Popa

Making an IP webcam work with your ODROID is a breeze

There have been some articles in the past year detailing how to set up your ODROID with a webcam to perform all sorts of interesting tasks, from detecting fires (http://bit.ly/2cviz9K) to augmenting reality (http://bit.ly/2cV74eA), and even enforcing home security (http://bit.ly/2dsqnen). For me, all I wanted was a webcam that worked over the Internet. A typical off-the-shelf IP webcam allows you to view the camera remotely in real time with sound through the Real Time Stream Protocol (RTSP), and usually has support for other features, such as grabbing a still image or controlling any pan or tilt functionality. This is often utilized to support robust home monitoring solutions, such as with a remote DVR to store your recordings, or to allow remote access on-demand. Android has plenty of apps that manage all of these needs for you, but we'll focus on Linux instead, because you might want to use your ODROID for other Linux-based tasks as well. By the end of this article, you will learn how to grab images from your webcam over the web, view real time streams with sound, and record your stream.

Setup the camera

Most modern cameras are supported under Linux with the generic "uvc" driver. The driver exposes several new devices to your Linux machine when a webcam is plugged in. For example, you may see a /dev/video0 Video4Linux interface, a new input device in ALSA, and maybe a button that acts like a HID keyboard. By installing the v4l-utils package, you can list the supported modes of your camera. An example listing for HardKernel's 720p webcam is at http://pastebin.com/L1VwZZFs.

```
$ sudo apt-get install v4l-utils
$ v4l2-ctl --list-formats-ext
```

You may notice that most cameras can output in YUV (uncompressed mode) with lower frames per second, or in MJPEG (compressed mode). High-end cameras can also output H264 video that is encoded directly inside the camera. This tutorial assumes you have a MJPEG-compatible camera available, but would like to view H264 streams from your system.

The v4l2-ctl utility also allows you to list and change some of the camera's parameters, such as the brightness, contrast, or gamma, which is useful if you don't have optimal lighting conditions. You can list these parameters with the following command:

```
$ v4l2-ctl --list-ctrls
```

If your camera doesn't expose a /dev/video0 pseudo-file, but

you can grab images with a custom API, you can use v4l2loopback (http://bit.ly/2cxa6rc) to feed your data to a virtual /dev/videoX device so that you can read it with standard tools.

Getting still images

Now that the camera is up and running, the first task is to grab images from it, whether to be saved on the local disk, or to be viewed remotely. Even if the task seems simple and there are several tools to help you do it, the details are important. Tools like uvccapture or streamer can do the job, but I found that in practice both suffer from the following problems:

Grabbing a picture initializes the camera and it can take a variable amount of time to complete, sometimes up to 30 seconds.

Pictures from these tools are usually dark because the camera hasn't had enough time to balance the light level. A streamer can compensate this by "recording" for a specified time, such as 1 second, before snapping the picture.

Sometimes the camera may return incomplete frames, such as where only the top part is visible.

Also, if you are using the camera for something else, like live streaming or motion detection, the tools can't connect to /dev/video0 to grab still images while recording, so you may also have a need for multiplex access to the camera.

The best tool for the job needs to have exclusive access to the video device while allowing other tools to grab images and video at the same time. Also, it needs to keep the camera active while taking images so that it compensates for darkness. For me, this miraculous tool was mjpg-streamer (http://bit.ly/2d2qSvQ). To install it under /usr/local, follow these steps:

```
$ git clone https://github.com/\
jacksonliam/mjpg-streamer.git
$ cd mjpg-streamer/\
mjpg-streamer-experimental
$ sudo apt-get install \
cmake libjpeg62-dev
$ make
$ sudo make install
```

It's best to test mjpg-streamer before enabling it at startup. The program has a configurable number of inputs (cameras) and several output settings. It can run as a HTTP server, output to a file on your local disk, or output to a UDP/RTSP streams. In my tests, the RTSP function was not reliable and did not work with any RTSP clients, which is likely as the RTSP protocol does not support MJPEG data streaming in standard implementations. In this tutorial, we will use it as a HTTP server and use other processes to read from mjpg-streamer.

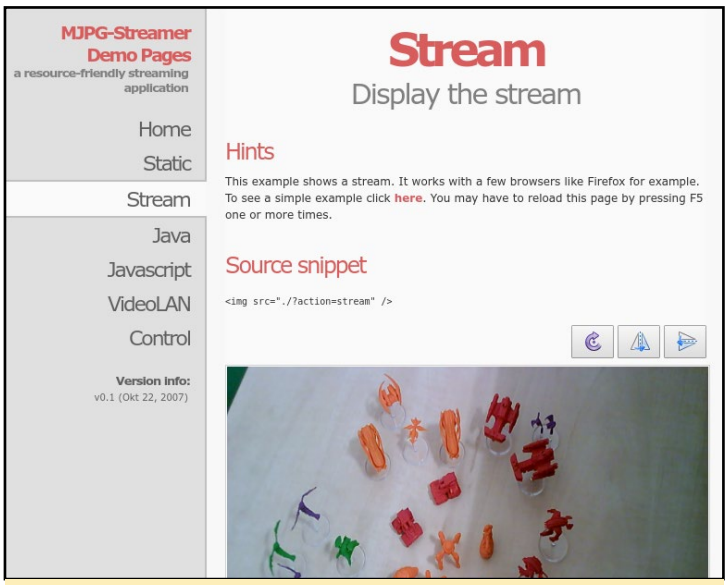
To start mjpg-streamer as a web server with authentication and read from the first camera, run the following command:

```
$ sudo /usr/local/bin/mjpg_streamer  -i 'input_uvc.
so -r 1280x720 -m 50000 -n -f 25 -d /dev/video0' -o
'output_http.so -p 8090 -w /usr/local/share/mjpg-
streamer/www/ -c odroid:odroidpass'
```

This command is complex, so let's explain what all the switches do. "-i" specifies input plugin, which is input_uvc.so (grabbing from a UVC camera). Next comes the camera's desired resolution, and "-m" specifies the minimum size of the input. I've set this to 50kB, so mjpg-streamer will drop any jpeg frames smaller than that (720p frames are around 120kB in size). This is a good thing, because sometimes the camera starts outputting incomplete frames, which are not useful images. This however also has the side effect of not capturing anything in low light conditions, as the frames are mostly dark and the jpeg compression reduces them to under 50kB. You will need to tune this parameter according to your input resolution.

The "-n" parameter disables dynamic controls in the UVC driver, while "-f" specifies the input framerate. "-d" points to the video device (/dev/video0 by default). On the output side of the equation, we use output_http.so module on port "-p" 8090 and serving HTTP files from the directory pointed to by "-w". You can optionally add password protection with the "-c" parameter and specifying the username:password combination. There is detailed usage information at http://bit.ly/2dbB97p and http://bit.ly/2dbALWx.

Once you successfully start mjpg_streamer as an HTTP server, you will be able to access it with a browser at http://<your-odroid-ip>:8090/. You will be asked for your user/password combination and be presented with the demo page, as shown in Figure 1. You can, of course, create your own page, but the demo page gives you all the necessary information to access the camera.



MJPEG Streamer web interface featuring some action figures

You can grab a still image from your ODROID using the following command:

```
$ sudo apt-get install curl
$ curl -s -f -m 5 http://odroid:odroidpass@odroid-
ip:8090/\
?action=snapshot > /tmp/snapshot.jpeg
```

You can use this together with crond to take pictures at a set interval of time. You can also use the timestamp as a filename or use a tool like montage to add the time as a watermark on top of the image. Here is a small script that snaps pictures in a specific directory of your ODROID's local disk and adds the date and time: `http://bit.ly/2d2fstx`. Additionally, you can use ffmpeg in a script like this to combine all these pictures in a video for easier later viewing: `http://bit.ly/2cOzXqY`.

Sample snapshot with the timestamp superimposed

To get an MJPEG video stream from the camera, which is essentially a sequence of JPEG images, you can run the following command:

```
$ vlc http://odroid:odroidpass@odroid-ip:8090/\
?action=stream
```

If all is well, and you are getting an image, it's time to add a systemd startup script for mjpeg_streamer. Create a file called /etc/systemd/system/mjpg_streamer.service with the contents downloaded from `http://bit.ly/2dbCPxO`. To activate the service, type the following commands:

```
$ sudo systemctl enable mjpg_streamer.service
$ sudo systemctl start mjpg_streamer.service
```

To check that the service is running, you can query systemd:

```
$ sudo systemctl status mjpeg_streamer.service
```

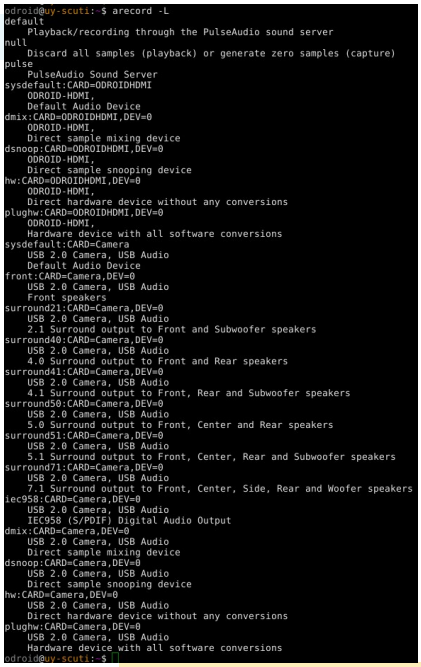
Getting videos

Motion JPEG is widely supported in all browsers, so that's a good thing, but it doesn't support sound, and compression is rather poor. The bit rate of 25FPS on a 720p MJPEG stream is around 13Mbps, which may be high for typical Internet usage. In order to get videos with sound, we'll need to multiplex the MJPEG stream with a sound stream from the camera's microphone into a supported media format.

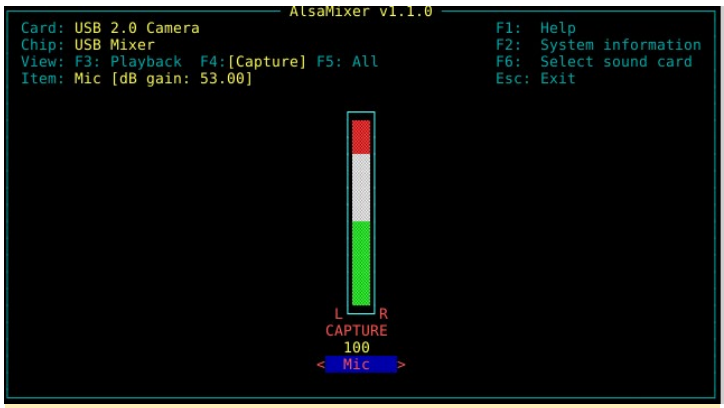
Since we know how to get the video stream, let's concentrate on the microphone. You can list the current devices supported by ALSA in your system with the "arecord -L" command. For the ODROID webcam, you should see several inputs relating to a USB 2.0 camera with varying capabilities, as shown in Figure 3. We will need the name in order to configure it in ffmpeg later on (in our case we'll use the last one - plughw:CARD=Camera,DEV=0).

```
$ arecord -L
```

Before we start recording, we need to check that the microphone is unmuted and is at an acceptable level. I really like the microphone on Hardkernel's 720p webcam, since it has adaptive gain so that I can hear whispers in a room followed by kids shouting, all without going deaf in the process. To tune the volume we will use alsamixer. First, press F6 to select the sound card and use F4 to go to the Capture tab. Use your arrow keys to adjust the audio level (I use it at maximum).



A listing of audio devices



Alsa mixer showing the audio level

We can now build our ffmpeg query that grabs a video stream from MJPEG Streamer, adds audio from ALSA, and produces a file on disk of that combined stream:

```
$ sudo apt-get install ffmpeg
$ ffmpeg -framerate 5 -f mjpeg -i 'http://odroid:odro
idpass@127.0.0.1:8090/?action=stream' \
-f alsa -i plughw:CARD=Camera,DEV=0 -acodec \
libmp3lame -c:v libx264 -preset ultrafast \
-r 5 -pix_fmt yuv420p -b:v 1500k \
-async 1 myvideo.mp4
```

The command above specifies that the input frame rate should be 5 FPS and that the input is the mjpeg stream from the address above. The "-f" parameter specifies that you should use ALSA for audio from the following device listed. Audio should be encoded with mp3lame and video with h264 using the ultrafast preset and an output frame rate of 5 FPS. The video bandwidth is limited to 1500 kbps, otherwise ffmpeg starts calculating what's best and you can't really do real-time encoding. The async option tries to synchronize video and audio, but drifts often occur anyway. Finally, the last parameter is the output filename that we want to write to.

With an ODROID-C2 you can (almost) do software encoding up to 10 FPS at 720p in real-time, but audio gets garbled, and your safest bet is to keep the framerate low. I have compiled an optimized version of ffmpeg for the C2 using the "-march=armv8-a+crypto+crc+fp+simd -mtune=cortex-a53" flag, but there was no noticeable change in encoding performance with this optimized version. Depending on your needs, this may or may not be acceptable. If you want a high frame rate, you need to reduce the resolution or switch to a XU4 where work has been done to support hardware encoding in the mainline kernel. You can read more about this at http://bit.ly/2cxbMkK.

My best results were with mjpg_streamer set to 640x480 and with ffmpeg recording at 10fps and with a video bandwidth of 1Mbps. Curiously, going lower than this results in poorer throughput at around 6 FPS. In case you hear choppy sound in your recordings, it means that ffmpeg can't keep up with the imposed frame rate. As far as I've seen, if you try to record to a higher framerate than what ffmpeg can do in real time, you will get choppy audio. Worst of all, the encoding performance depends on system load, so on higher loads you'll get a lower FPS in real time. To see a few recipes that I have tried, and also to see how to record audio only, consult this cheat sheet at http://bit.ly/2cvjBlO.

I also redid the tests after Hardkernel pushed their new overclock boot.ini settings with the C2 running at 1.75GHz and 4 cores. Using this, I was able to get stable sound at 720p with 8 FPS (instead of 5), and 15 FPS with a resolution of 640x480, which is nice. I wasn't able to test performance at higher frequencies with less cores due to too much instability, but I expect things to improve over time. Also, if you increase the RAM frequency to 1104 MHz, you can expect to gain 1 to 2 FPS.

If you want to bypass mjpg_streamer completely, you can also read directly from /dev/video0:

```
$ ffmpeg -r 5 -f v4l2 -video_size 640x480 \
-i /dev/video0 -f alsa \
-i plughw:CARD=Camera,DEV=0 -acodec \
libmp3lame -c:v libx264 -preset ultrafast \
-r 5 -pix_fmt yuv420p -b:v 1000k \
-async 1 myvideo.mp4
```

In fact, ODROID Forum contributor @crashoverride just recently released a library and a test program that allows you to do hardware encoding of H.264 on the C2 at full FPS, but it needs raw access to your camera, so mjpg_streamer must be disabled. Work is being done to further improve this method, so make sure to check out its support thread for updates at http://bit.ly/2dcQDJn, as updates can happen quickly.

Streaming RTSP on demand

The main use of an IP camera is to be able to view the video stream on demand. Ideally, it should be viewable by multiple concurrent users at the same time. In order to do this, we will use ffserver to create a RTSP stream that plays on demand.

The Real Time Streaming Protocol (RTSP) is a protocol similar to SIP that handles signalling and media transport between a client and a server. Usually, signalling is done on TCP port 554 and the data streams over UDP with the client and server negotiating a suitable port. However NAT and firewall environments sometimes interfere with this negotiation, so there is also a way to transport the data over TCP interleaved with control traffic. This transport method will be used for our tests.

The ffserver application provides a way of serving RTSP client requests based on ffmpeg video feeds. It is part of the ffmpeg package, so you already have it installed if you've followed the tutorial up to this point. To start the server, you'll need a suitable configuration and a systemd startup script. The configuration needs to be saved to /etc/ffserver.conf and you can get one from http://bit.ly/2cYWPcq.

If you browse through the configuration, it sets up a listener on RTSP port 554, defines a feed called mjpg-streamer.ffm and ties it to an output stream called live.h264.sdp. The ffserver application allows you to set up different output formats, but for this example it will pass through the input stream, which will be already h264.

To start ffserver at startup, you will need to add the follow-

ing systemd service to the file /etc/systemd/system/ffserver.service:

```
https://github.com/mad-ady/odroid-webcam-scripts/blob/master/ffserver.service
```

To enable it and see its status do:

```
$ sudo systemctl enable ffserver
$ sudo systemctl start ffserver
$ sudo systemctl status ffserver
```

At this point, you have an RTSP server listening for requests, but no video is being processed. To start a video feed, you will need to run ffmpeg like this:

```
$ /usr/bin/ffmpeg -loglevel 8 \
-r 5 -f mjpeg -i 'http://odroid:odroidpass@127.0.0.1:8090/?action=stream' \
-f alsa -i plughw:CARD=Camera,DEV=0 \
-acodec libmp3lame -c:v libx264 \
-preset ultrafast -r 5 \
-pix_fmt yuv420p -b:v 1500k \
-async 1 -x264-params keyint=30:no-scenecut=1 \
-vf "drawtext=fontfile=/usr/share/fonts/truetype/dejavu/DejaVuSans-Bold.ttf: text='Webcam feed %{localtime\\:%F %T}': fontcolor=white@0.8: x=7: y=5" \
-override_ffserver http://localhost:8099/mjpg-streamer.ffm
```

Before you panic at the complexity of this command, note that it's similar to the one you've seen before with the simple addition that we add an overlay text in the top left corner with the current date and time, just like the "professional" IP webcams! The ffmpeg application writes the output to ffserver, specifying the feed name.

You should now be able to connect with a RTSP viewer and enjoy your video feed. If you're testing from your Android smartphone, you can try RTSP Viewer, available at http://bit.ly/2cvl0J8:

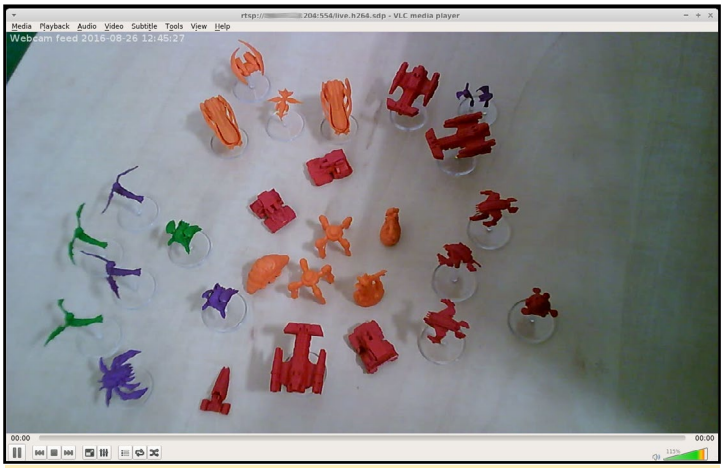
```
$ vlc rtsp://odroid-ip:554/live.h264.sdp
```

To make things more permanent, you can add the "https://github.com/mad-ady/odroid-webcam-scripts/blob/master/ffmpeg.service" systemd ffmpeg service file into /etc/systemd/system/ffmpeg.service. To enable it and see its status, type the following commands:

```
$ sudo systemctl enable ffmpeg
$ sudo systemctl start ffmpeg
$ sudo systemctl status ffmpeg
```

The XU4 with mainline the kernel can already do hardware encoding, if you have one around, so if your cameras are connected to other hardware like C2, you can run ffserver on the XU4 and read the MJPEG stream from the C2 over the network (hopefully wired), get mp3 audio from an ffserver running on the C2, and transcode video on the XU4 before presenting it to the viewer. When my XU4 is operational, I plan to offload transcoding to it and will post the changes on the support thread in order to improve support for multiple cameras/streams.

Improve idle performance



RTSP streaming with sound

The video feeds are expected to be up at all times, which means ffmpeg must be transcoding even if there is no viewer connected. This may be fine if you expect to have a lot of viewers connected simultaneously, but if you intend to connect rarely (e.g., 5 minutes/day), it's not worth it to have the stream transcode in the background when not being used. It would be best if we had a system that could trigger the start of the video stream when a viewer connects and could trigger the stop of the stream when all viewers disconnect. I wrote the ffserver-trigger script just for this scenario.

The script runs in the background and continuously runs a tail -f command on /var/log/syslog. It will pick up messages from ffserver such as "PLAY live.h264.sdp", check if the stream is already running, and start it if not. It will also look for stop messages such as " RTP/TCP" and stop the stream if necessary. It logs its actions under syslog as well, for added convenience. Note that this trigger is customized for a single stream and follows the naming convention used in the article. It might need tweaking if you want to use it for other setups.

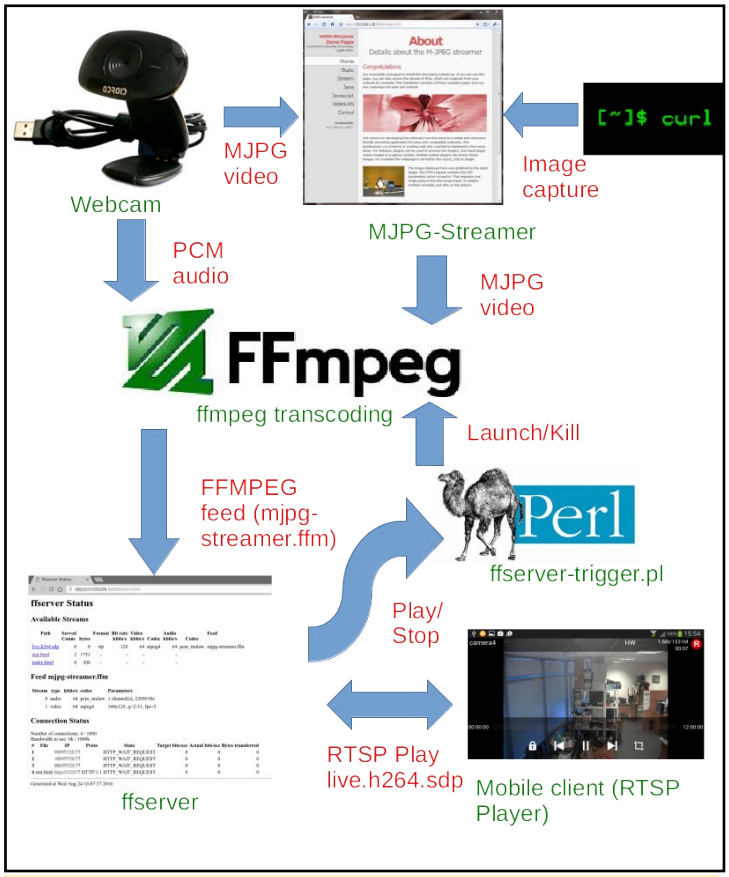
To install ffserver-trigger, type the following commands:

```
$ sudo apt-get install libfile-tail-perl
$ sudo perl -MCPAN -e 'install Linux::Proc::Net::TCP'
$ sudo wget -O /usr/local/bin/ffserver-trigger.pl \
https://raw.githubusercontent.com/mad-ady/\
ffserver-trigger/master/ffserver-trigger.pl
$ sudo chmod a+x /usr/local/bin/ffserver-trigger.pl
$ sudo wget -O /etc/systemd/system/ffserver-trigger.
service \
https://raw.githubusercontent.com/mad-ady/\
ffserver-trigger/master/ffserver-trigger.service
$ sudo systemctl enable ffserver-trigger
$ sudo systemctl start ffserver-trigger
$ sudo systemctl status ffserver-trigger
```

Since you're using the ffserver-trigger now, you should disable the ffmpeg service so that it doesn't start automatically on boot. Instead, it will be started by ffserver-trigger when needed.

```
$ sudo systemctl disable ffmpeg
```

Figure 6 shows the entire workflow.



Our streaming pipeline

If you want to be able to also record your stream to a file, you can connect to it as a regular RTSP viewer and dump it to a file without transcoding. This is advantageous, since you can do it even while other clients are connected without interrupting their experience:

```
$ ffmpeg -i rtsp://127.0.0.1:554/live.h264.sdp \
 -acodec copy -vcodec copy rtsp-recording.mp4
```

In terms of video processing delays, mjpg_streamer adds about a 1 second delay, while ffmpeg + ffserver will add between 2-3 extra seconds. At those speeds, your experience will not be real-time, and not suitable for remote controlling a robot, but should be good enough for remote viewing.

Troubleshooting Tips

Question: I'm unable to get images from mjpg_streamer/ffmpeg seems stuck.
Answer: Check the value of -m parameter, and lower it to fit your needs.

Question: How do I fix my audio being out of sync of choppy?
Answer: Try 640x480 @ 10fps, or reduce the frame rate in ffmpeg.service.

Question: Why does stopping a RTSP stream stops all connected clients?
Answer: Sometimes ffserver will crash with a segfault when a client stops. It gets restarted automatically by systemd, but will disconnect all clients.

Question: Pressing Play as the first connected client does not start the RTSP stream when using ffserver-trigger, why is that?
Answer: This is a known issue. The RTSP stream will timeout in about 10s before ffserver manages to send data back to the client. Press Play again after the timeout. If a client connects when a stream is active, this issue doesn't happen. The trigger script has a 20 second cooldown period in which it will ignore stop requests after a stream start to mitigate this.

Question: Sometimes connecting to a stream doesn't work, and ffmpeg seems stuck. How do I fix this?
Answer: The cause is mjpg_streamer. It sometimes gets stuck and needs to be restarted. There are two lines you can uncomment in ffserver-trigger.pl to restart it automatically when ffmpeg is restarted to avoid this.

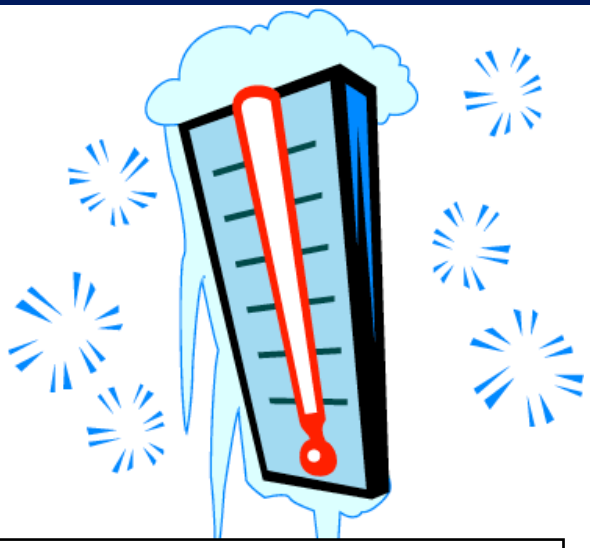
Question: But wait, an off-the-shelf webcam has pan and tilt support. How do I add those to my camera?
Answer: You can add them with some motors and PWM pins or an Arduino (`http://bit.ly/2diWcKh`).

If you run into other problems, or if you find better ways to achieve this, feel free to let me know on the support thread for this guide at `http://bit.ly/2d2j6DH`.

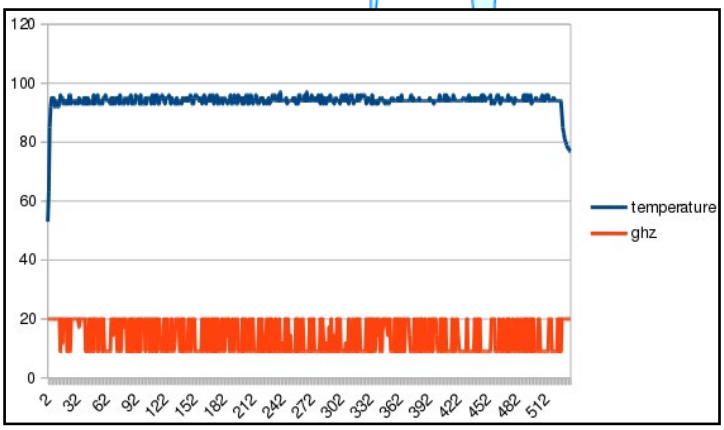
ODROID-XU4 COOLING TESTS

DISCOVER THE BEST COOLING FOR YOUR NEEDS

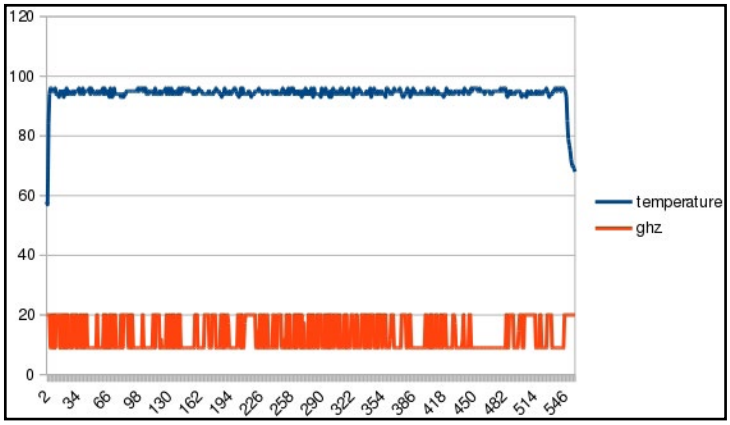
by Bo Lechnowsky



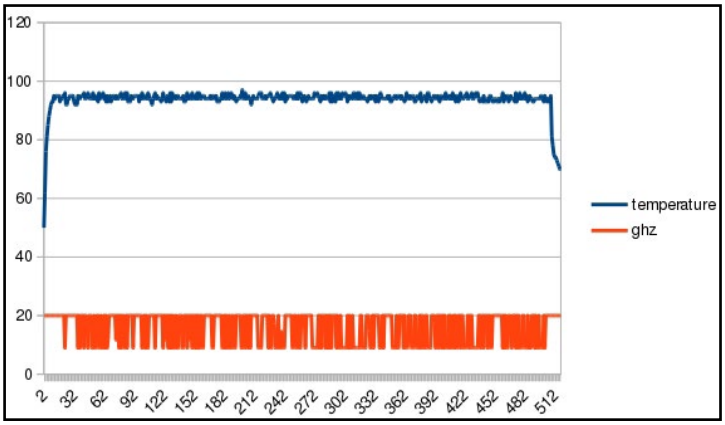
I ran several temperature tests using an ODROID-XU4 with different types of cooling systems. The goal was to find the most effective way to cool the device while under heavy load. In the graphs below, the temperature is indicated by a blue line, and the speed of the XU4 is marked in orange. The device will throttle its speed based on temperature, and the best situation is where the temperature remains under 95 degrees Celsius indefinitely which minimizes throttling, keeping the effective clock speed as close to 2 GHz as possible. Each test ran for over approximately 5-6 minutes, which is indicated on the X axis, and the temperature tends to remain below 100 degrees Celsius, which is indicated on the Y axis.



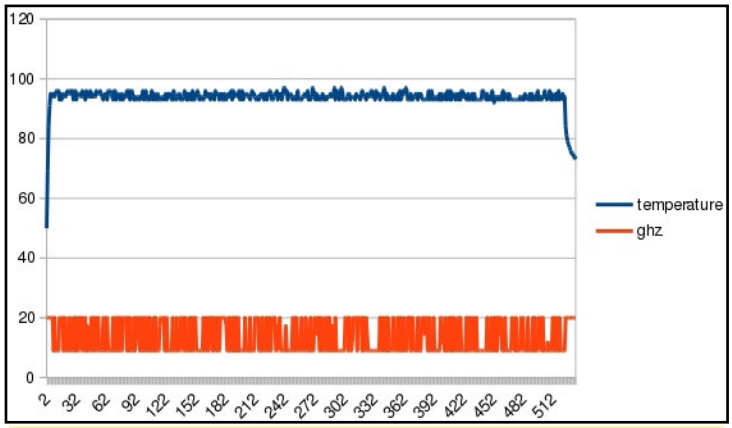
Passive Northbridge Heatsink (Gold)



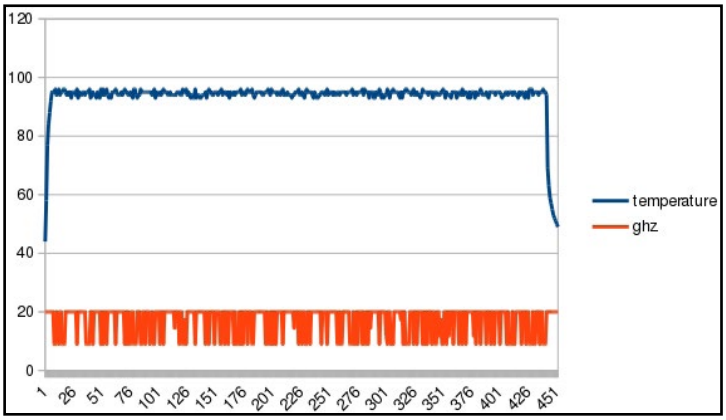
XU4 Default Active Cooler



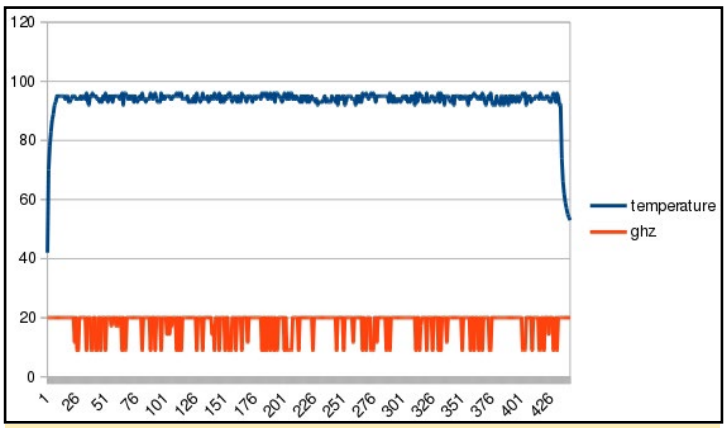
Passive Shapedmedia Aluminum Enclosure



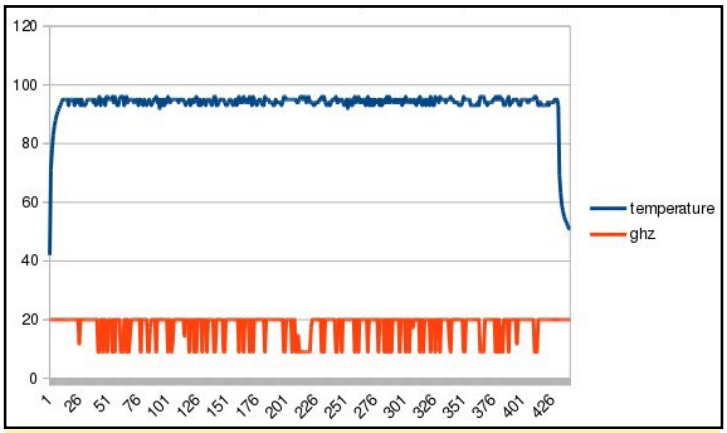
Passive Northbridge Heatsink (Blue Zalman)



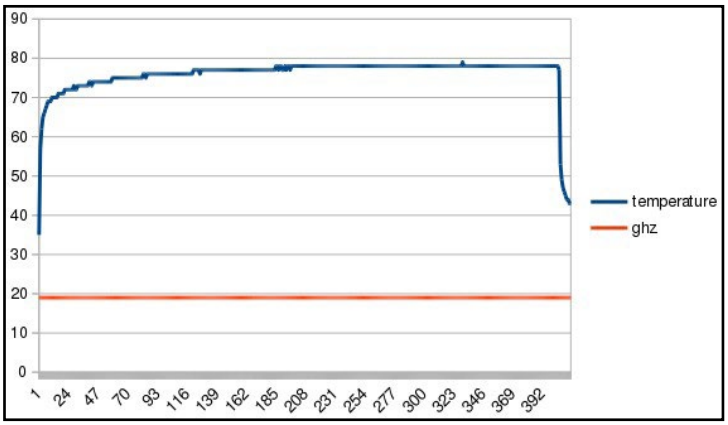
Passive Northbridge Heatsink (Blue Zalman) ventilated by 120mm case fan



Active Northbridge Heatsink (Gold) ventilated by 40mm Noctua 5VDC fan



Passive Northbridge Heatsink (Gold) ventilated by 120mm case fan



Passive Shapedmedia Aluminum Enclosure

Conclusion

The best way to cool the ODROID-XU4 is to use a thermal compound in conjunction with a 40mm Noctua 5V DC fan and an active Northbridge heatsink, which may be purchased at `http://bit.ly/2cBeTGm`. It keeps the temperature below 80 degrees Celsius, and allows the device to run at 1.9GHz effectively without throttling. The most effective passive cooling methods was the Shapedmedia case, available from Ameridroid at `http://bit.ly/2d4YCMH`, which kept the ODROID-XU4 at around 95 degrees Celsius.

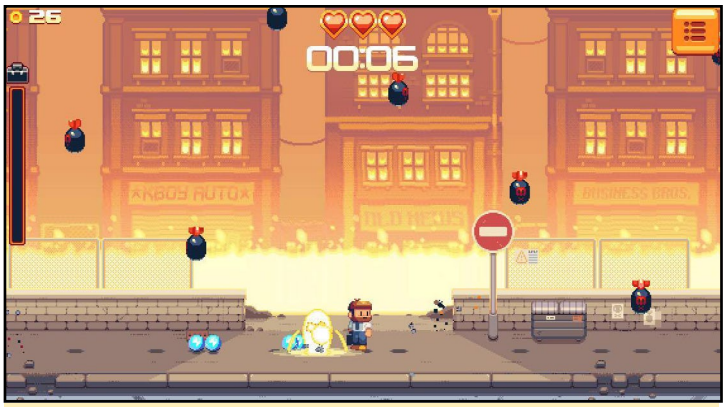
ULTIMATE BRIEFCASE

GET READY TO RUMBLE IN A FRANTIC SURVIVAL GAME

by Bruno Doiche

Well, it is a typical day, you are walking around with your briefcase, and by a little mistake you slip on a banana peel, escaping from the first of many, many, maaaany bombs that are being thrown towards you!

What happened? Will it ever end? Is there a reason for this violence against your person? Dodge the bombs, get your power ups, unlock items, recruit other characters from a really endearing ensembl,e and get ready for a super entertaining game. If you are good enough, you may very well discover that there is more to this story then meets the eye!



Who among us ever took the time to survive an endless bombing?

You can use items to survive longer to see more of the game story

`https://play.google.com/store/apps/details?id=com.nitrome.ultimatebriefcasew`

INDUSTRIAL AUTOMATION

REMOTELY MONITOR MODBUS REGISTERS USING AN ODROID-XU4

by Joel Duncan

The field of Industrial Automation is not known to adopt mainstream technologies at a fast pace. This is partly due to the large monopoly held by the three main players: Siemens, Allen Bradley and Wonderware. For various reasons, they do not aggressively adopt innovations, such as building native industrial web applications. At Bubble Automation, we realized this shortcoming. Most clients who wanted remote monitoring capabilities of their sites were stuck using inefficient proprietary add-ons. Some of these add-ons required high license fees and maintenance costs, or insecure TeamViewer/VNC connections needing third party tools to be installed on the client's systems.

Project goals

We wanted to build a modern secure native web application that required no special browser or PC plugins. While handling live data, the application was required to work on any computing device, including, smartphones, tablets, netbooks and desktops. The design called for no dependency on unnecessary platform specific applications. To ensure high levels of security, the web server hardware had to be on the customer's premise, be small enough to be installed in a control panel, and be robust enough to survive industrial conditions.

Choosing the framework

NodeJS was the first framework that we tested. It seemed promising, but at the time of evaluations, was not the most stable or well-supported platform. The biggest issue was connecting to a database using their experimental database module. After trial and error with various frameworks, we settled for a pure Python solution, as we had staff experienced in advanced Python techniques and use of its vast array of special purpose modules.

Selecting the hardware

After a very brief experience using current industrial grade single board computers (SBCs), it was immediately obvious that they weren't suitable. Most use out-dated Intel Atoms in large noisy enclosures. This sent us on the path of today's competitive credit sized PC market, and here are a few that we tried:

- Raspberry Pi 2 Model B
- Raspberry Pi 1 Model B
- ODROID-U3
- ODROID-C1
- ODROID-XU4

Each board was extensively tested running our Python framework with unrealistic loads. The board that stood out was the ODROID-XU4 which was clearly superior all around, as shown in the benchmark results in Figures 1-4.

Figure 1 - Dhrystone results are an index relative to a SPARCstation 20-61, rated at 10.0

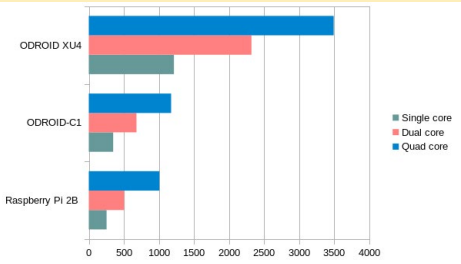


Figure 2 - Whetstone results are an index relative to a SPARCstation 20-61, rated at 10.0

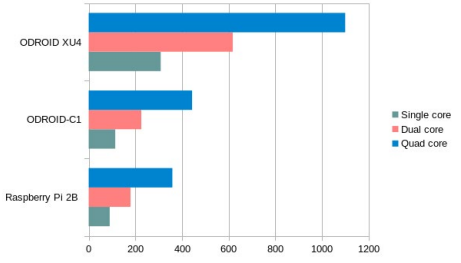
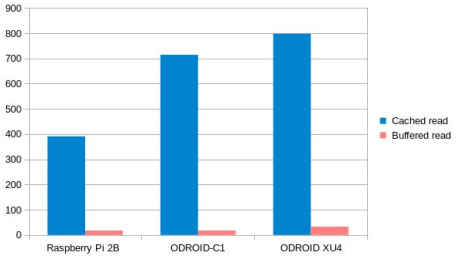


Figure 3 - Hdparm results are in megabytes per second



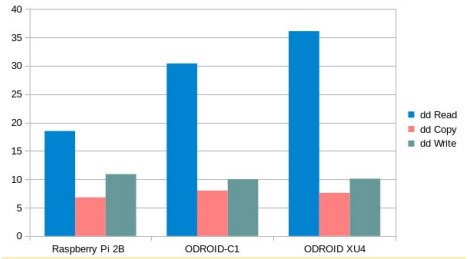


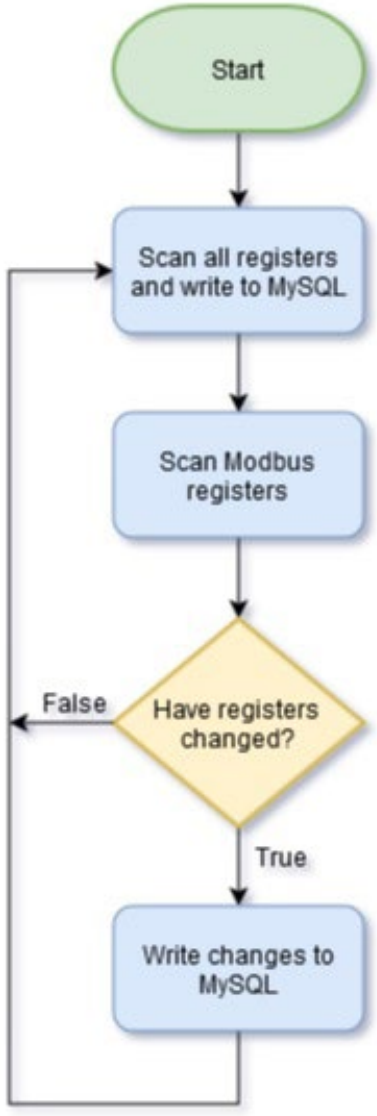
Figure 4 - DD results are in megabytes per second

This board had far better CPU and network performance, but most importantly, exhibited much higher I/O speeds and bandwidth, which was critical for our MySQL based database.

Architecture & Software Design

The basic starting point was to create a daemon that could read Modbus registers over TCP/IP from the field, as shown in Figure 5. This later grew to a context-sensitive event-based system that could translate field events into alarms, live trending, historical graphs, event logs and notification emails.

Figure 5 - Application flowchart



Providing this information to the client proved to be a difficult task. Displaying real-time information on a web page using pure HTML5 and no plugins, has always been challenging. Using techniques such as long-polling would only lead to crashing the browser, as there was just far too much information that needed to be fed into the browser at relatively high speed, at around 1-second scan intervals.

We noticed that NodeJS was such a good candidate for Real Time data solutions, due its efficient integration with Websockets, which is a technology providing full-duplex communication over a TCP connection. Thankfully, we came across a powerful PHP Library that provided this functionality. For this reason, we could build all server-side components in PHP using Twitter Bootstrap to provide a simple interface with responsive design at its foundation.

A LEMP stack (Linux, Nginx, MySQL, & PHP) is used to provide a robust base with speed, stability and reliability for our PHP, HTML and Javascript front-end. The main reasons for using Nginx over Apache is its better utilization of multicore processors assigning a worker per core and its better security track-record over Apache.

Security

Due to the nature of our industry, building our application to be secure by design was critical. A lot of time was spent creating a secure login not vulnerable to SQL injection, session hijacking, cross-site scripting and brute force attacks. To top this off, we created a hardened minimal Linux image to use on our servers this is kept up to date with all of the current security patches. We work strictly on a non-control basis, which means that in the unlikely event that our software is compromised, there is no way the intruder could damage the site. Our software only monitors the state of the field and does not affect the control process.

For comments, questions and suggestions, please visit the original post at http://bit.ly/2cp6tzj.

Figure 6 - Front-end screenshot

NVIDIA GAME STREAMING ON THE ODROID-C2

PLAY MODERN GAMES ON YOUR ODROID

by @khaine

NVIDIA GAMESTREAM™

NVIDIA's GameStream technology lets you stream games from a GeForce-powered Windows PC to another device. It only officially supports NVIDIA's own Android-based SHIELD devices, but with a third-party open-source GameStream client called Moonlight, you can stream games to your ODROID.

PC installation

First, you'll need to set up NVIDIA GameStream on your Windows PC, and you'll need to be using an NVIDIA video card for this to work. If you don't have the GeForce Experience software installed, you'll need to download and install it from NVIDIA at http://bit.ly/1kIWAdz. Then, launch the "GeForce Experience" app from your Start menu. Click the "Preferences" tab at the top of the GeForce Experience window and select the "SHIELD" category. Ensure that the "Allow this PC to stream games to SHIELD devices" box is checked.

If you want to add any custom games that GeForce Experience didn't automatically find, you can add them to the Games list under Preferences -> Shield. You can actually add any program here, even desktop programs.

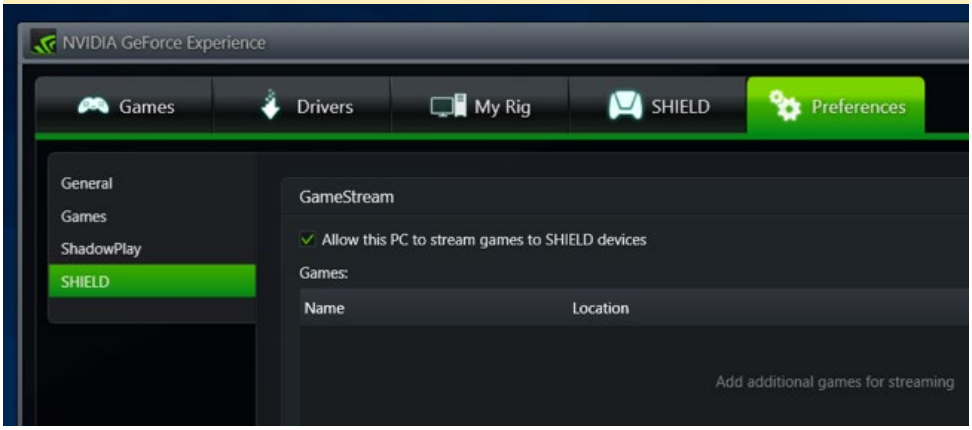
Moonlight installation

1. Install the ODROID-C2 Debian Jessie image from http://bit.ly/2cj6V6F and boot it.

2. Update the image with the following command, which could take a while:

```
$ sudo apt-get update && \
  apt-get-upgrade && \
  apt-get dist-upgrade
```

Figure 1 - Setting up NVIDIA Game Streaming on the PC



3. Install Moonlight:

```
$ sudo apt-get install moonlight-
embedded
```

4. Install PulseAudio (newer version of pulseaudio is reported to have less audio delay):

```
$ apt-get install -t \
  jessie-backports pulseaudio
```

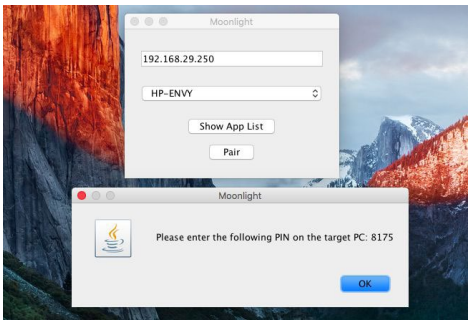
5. Reboot the computer, at which point Moonlight should be working both on H.264 and H.265.

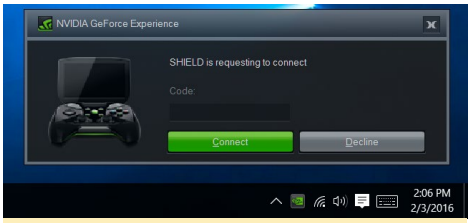
6. If you also want to use Kodi, run the following command and install both Mate Desktop and Kodi:

```
$ sudo setup-odroid
```

9. If you also want to autostart Kodi, you can go to the "Startup Applications" section of the Control Center and add Kodi.

10. To start Moonlight directly from Kodi, you can install Luna from http://bit.ly/2cWy3sD. If you are only using Steam, you could create some automation by yourself. For example, you could create a Steam streaming systemd unit by creating a file at /etc/systemd/system/steam.service with the following contents:





Figures 1 and 2 - Connecting to the NVIDIA Game Stream using the generated PIN

```
[Unit]
Description = Steam Streaming
After = systemd-user-sessions.
service network.target sound.
target
Conflicts = kodi.service
OnFailure=kodi.service

[Service]
User = odroid
Group = odroid
Type = simple
ExecStart = /usr/bin/moonlight
stream -app Steam -60fps -1080
STREAMING_MACHINE_IP
ExecStop = /usr/bin/moonlight
quit STREAMING_MACHINE_IP
ExecStopPost = /usr/bin/sudo /
bin/systemctl start kodi
```

Then, add a Kodi shortcut to the System.Exec file at /home/odroid/steam.sh that points to steam.sh script:

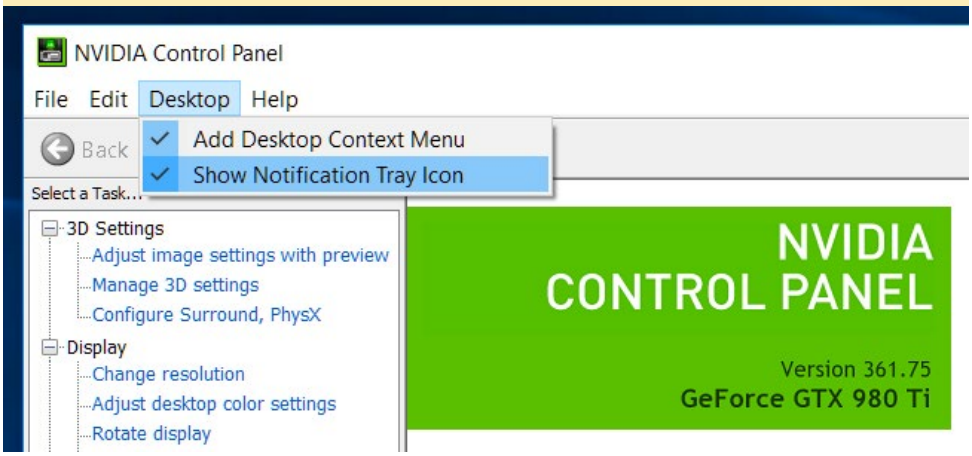
```
#!/bin/bash
sudo /usr/bin/nohup /bin/system-
ctl start steam &
```

Now you should have both Kodi and Moonlight working with easy switch between them. To connect to your PC, the Moonlight app will give you a PIN. Enter it in the "SHIELD is requesting to connect" pop-up that appears on your PC, and your devices will be paired.

If you don't see the PIN request dialog, open the NVIDIA Control Panel application on the Windows PC, click the "Desktop" menu, and select "Show Notification Tray Icon." The next time you attempt to pair your devices, the PIN pop-up will appear. For whatever reason, the PIN pop-up is tied to this system tray icon, and it just won't appear if you haven't enabled it.

For comments, questions or suggestions, please visit the original post at http://bit.ly/2cYgG74, or check out the official documentation at http://bit.ly/1skHFjN.

Figure 3 - Show Notification Tray Icon selection



LINUX GAMING

SOME GREAT GAMES FOR THE ODROID-C2

by Tobias Schaaf

The ODROID-C2 is a great gaming device, and this month I'd like to highlight some of my game ports that are compatible with the C2. I recently updated several of them with new binaries and improvements. I recommend using my Debian Jessie 64-bit image, which already has my repository installed. If you are using a different Ubuntu or Debian distribution, type the following commands to access my software packages:

```
$ wget http://oph.mdrjr.net/meveric/\
sources.lists/meveric-jessie-main.list
$ wget -O- http://oph.mdrjr.net/meveric/\
meveric.asc | apt-key add -
$ apt-get update
```

Unfortunately, this might not work for all games and distributions, since some dependencies are not yet available on Ubuntu.

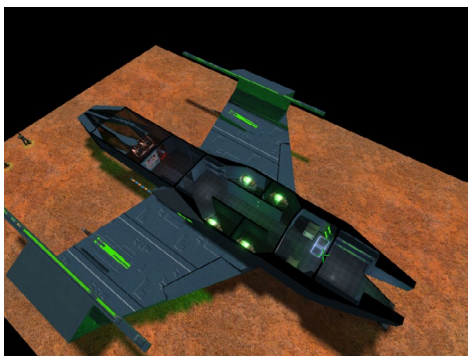
UFO: Alien Invasion

UFO: AI is a remake of the original XCom series in a more modern way, with much better graphics. Originally, @ptitSeb ported it to the Pandora, and I used his build to bring it to ODROID, but by now many of his changes were merged into the main project and I'm able to build the newest version of UFO: AI for ODROIDs. The game offers nice 3D graphics different languages and resolutions and very good music. It is similar to the original XCom game, but in a modern style.

To download and install UFO: Alien Invasion, type the following command:

```
$ sudo apt-get install ufoai-odroid
```

Figures 1 - 3 - UFO: Alien Invasion



SmokinGuns

SmokinGuns is a wild west shooting game based on the ioQuake (Quake 3) engine. It's fun in multiplayer and offers some nice game modes, such as defending a bank from bandits. It runs very smoothly on the ODROID-C2 even in high settings. You can check out a gameplay video https://youtu.be/RGI-YZf-BBfA.

Figures 4 - 6 - SmokinGuns

To download and install SmokinGuns, type the following command:

```
$ sudo apt-get install
smokinguns-odroid
```

CorsixTH

CorsixTH is a remake of the old classic game Theme Hospital, similar to OpenTTD. It is a very close rebuild to the original version with some improvements, like different resolutions. The Game is still a work in progress, but is far from being just an alpha version. You can play the original game just fine with lots of improvements, and it will get better and better over time as new versions are released. A gameplay video is available at https://youtu.be/rSN1p247J74. You need the original files in order to play the game. Version 0.60 was recently released, which added the following features:

- User campaigns: it is now possible to create a series of levels that play together just like the original game.
- In game map editor: a new map editor is available directly from the game menu.
- Drug price impact: patients will now react to the price you set for treatments. If treatments cost too much patients will opt to go home instead and this can affect your reputation.
- Variable spawn rate: the spawn rate will now take into account your hospital's reputation (after a date set in the level file.)
- Machine smoke is now visible when machines are close to worn out.
- Right-clicking the timer is now supported like in the original game for moving to affected patients
- Numpad support - The number pad can now be used to move around the map when numlock is off, or to type numbers when numlock is on, without side effects.

Figures 7 - 8 - CorsixTH

- The MP3 folder can now be unset from within the game. Previously after setting an mp3 folder it could only be removed by editing the config file by hand.
- The interface now uses the proper cursor for resizing rooms.
- Bins can now be placed in the hallways.

To download and install Corsix TH, type the following command:

```
$ apt-get install corsixth-odroid
```

YQuake 2

Yquake 2 is an open-source Quake 2 remake it even supports multi-player matches against other players. The game runs in full speed in 1080p resolution, and uses GLShim.

Figures 9 - 10 - YQuake 2

To download and install YQuake 2, type the following command:

```
$ apt-get install yquake2-odroid
```

There is also a high resolution pack available, which is shown in the screenshots. Happy fragging!

COMPILING ANDROID LOLLIPOP FOR THE ODROID-C2

USING LINUX MINT 18 OR UBUNTU 16.04

by Jörg Wolff

This guide details how to compile Android 5.1.1 Lollipop for the ODROID-C2 on a computer running either Linux Mint 18 or Ubuntu 16.04. The guide will build upon the official Hardkernel instructions for compiling Android. Before compiling Android Lollipop, there are a few software packages and build dependencies that must be set up. For Android Lollipop, the Java Development kit (JDK) OpenJDK7 is required. Since both Ubuntu 16.04 and Linux Mint 18 use OpenJDK8, OpenJDK7 must be installed from the Personal Package Archives (PPA) sources. To install OpenJDK 7, enter the following commands in a terminal window:

```
$ sudo add-apt-repository \
  ppa:openjdk-r/ppa && sudo apt-
get update
$ sudo apt-get upgrade && \
  sudo apt-get dist-upgrade
$ sudo apt-get install adb fast-
boot openjdk-7-jdk
```

It is not necessary to remove the default version of OpenJDK 8 that comes preinstalled. In fact, it is recommended to leave it installed as it may be needed for other programs such as Eclipse or Android Studio. Both versions can co-exist side-by-side, but before compiling Android, the Java version must be switched to OpenJDK7. This can be done with these commands:

```
$ sudo update-alternatives \
 --config java
$ sudo update-alternatives \
 --config javac
$ sudo update-alternatives \
 --config javadoc
```

The console output from the previous commands should look like this :

```
  Auswahl      Pfad
Priorität Status
------------------------------
---------------------------
  0            /usr/lib/jvm/
java-8-oracle/jre/bin/java
1082      automatischer Modus
* 1            /usr/lib/jvm/
java-7-openjdk-amd64/jre/bin/java
1071      manueller Modus
  2            /usr/lib/jvm/
java-8-openjdk-amd64/jre/bin/java
1081      manueller Modus
  3            /usr/lib/jvm/
java-8-oracle/jre/bin/java
1082      manueller Modus
```

For successful compilation of Android, there are several required packages that must be installed on the host computer (the computer doing the compiling). The packages can be installed with the following apt-get command:

```
$ sudo apt-get install adb fast-
boot git\
 ccache automake lzop bison
gperf\
 build-essential zip curl zlib1g-
dev\
 zlib1g-dev:i386 g++-multilib
python-networkx\
 libxml2-utils bzip2 libbz2-dev
libbz2-1.0\
 libghc-bzlib-dev squashfs-tools
pngcrush\
 schedtool dpkg-dev liblz4-tool
make\
 optipng maven python-mako py-
thon3-mako\
 python python3 syslinux-utils\
 Google-android-build-tools-in-
staller
```

These are the preparations for your

host computer. On my system, I had some build errors. If you have accidentally started to compile with a wrong JDK version, it is best to clean the git repo, or to redownload the source and start fresh. Since downloading can take 1-2 days, the cleaning option should be tried first. To clean the source, run the command from the top folder of the source tree. Additionally, the "out" folder should be removed:

```
$ repo forall -c \
  'git reset --hard ; git clean
-fdx'
$ rm -rf out
```

The next steps are well documented on Hardkernel's wiki page at http://bit.ly/2chF4Tu. When following these steps, don't forget to install the following toolchains, without which you will run into build errors:

- gcc-linaro-arm-none-eabi-4.8-2014.04_linux
- gcc-linaro-aarch64-none-elf-4.9-2014.09_linux

Repo sync failure

If you are dealing with old sources, a repo sync might fail. If that happens, then it is needed to force the sync:

```
$ repo sync --force-sync
```

And after cleaning and syncing, don't forget to switch to the master branch:

```
$ repo start s905_5.1.1_master
--all
```

You may also encounter this recurring error:

```
unsupported reloc 43
libnativehelper/JniInvocation.
cpp:165: error: unsupported reloc
43
```

After a lot of trial and error, I found a workaround on the xda-developers forums at http://bit.ly/2cCtfrp:

```
$ ln -sf /usr/bin/ld.gold /home/
(your account name)\
/(build source repository)/pre-
builts/gcc/linux-x86/host/\
(glibc version)/x86_64-linux/bin/
ld
```

Note that it needs to check the error messages, regarding which version of x86_64-linux-glibc2 is used. On my system, I was successful with the following command, which creates a symlink id:

```
$ ln -sf /usr/bin/ld.gold \
/home/joerg/odroid_c2/lollipop/\
prebuilts/gcc/linux-x86/host/\
x86_64-linux-glibc2.11-4.6/
x86_64-linux/bin/ld
```

Always make a backup first! Another error that may occur is the following:

```
public_api.txt:20: error 5
out/target/common/obj/PACKAGING/
public_api.txt:20: error 5: Added
public field android.Manifest.per-
mission.BACKUP
```

The solution is to type the following command:

```
$ make update-api
$ make -j4 selfinstall
```

Now you can follow the build steps from Hardkernel's wiki page:

```
$ export ARCH=arm64
$ export CROSS_COMPILE=aarch64-
none-elf-
$ export PATH=/opt/toolchains/\
gcc-linaro-aarch64-none-
elf-4.9-2014.09_linux/\
bin:$PATH
$ export PATH=/opt/toolchains/\
gcc-linaro-arm-none-ea-
bi-4.8-2014.04_linux/\
bin:$PATH
$ export JAVA_HOME=/usr/lib/jvm/
java-1.7.0-openjdk-amd64
$ export PATH=$JAVA_HOME/
bin:$PATH
$ source build/envsetup.sh
$ lunch odroidc2-eng-32
$ make -j4 selfinstall
```

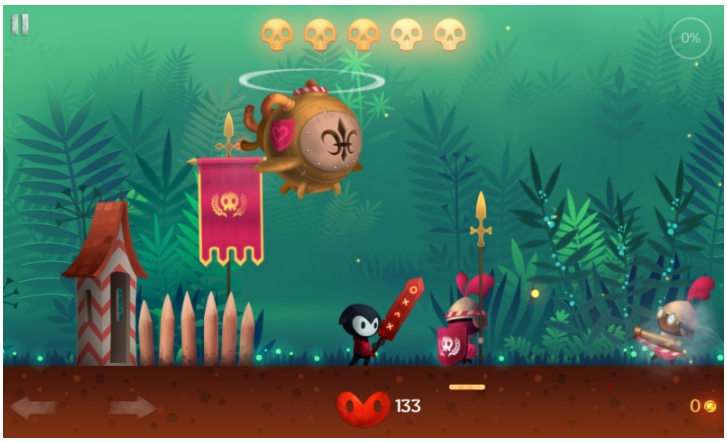
If you have installed the toolchain to another folder, as I normally do, you will need to supply the correct path to the toolchains. These steps should give you a working version of Android that is ready to be installed on an eMMC module or SD card.

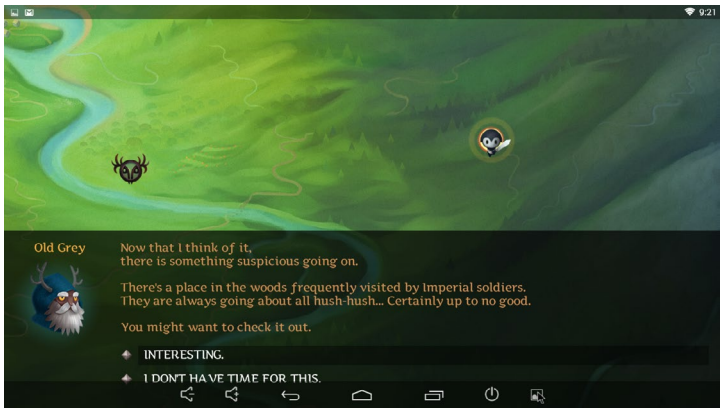
ANDROID GAMING

REAPER, TALE OF A PALE SWORDSMAN

by @synportack24

If you're looking for a great looking fun action video game for Android, Reaper is it! Reaper: Tale of a Pale Swordsman is an action and role-playing video game available for free from the Google Play store at http://bit.ly/1sIRjYd.



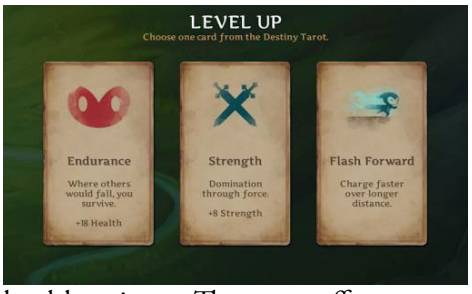


The game takes place in a land known as the "wilderness," where your character, the pale swordsman, has just awoken. The "wilderness" is a magical land full wild tribes and imperial guards, and various groups in-between. The pale swordsman works as a type of mercenary by going on quests and adventures for the different groups of the wilderness and his friend named Old Grey Beard. Later in the game, a new enemy arises which threatens not only the pale swordsman but the entire population of the wilderness.

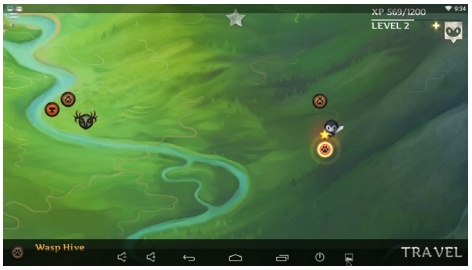
The gamer controls the swordsman and can choose the path that the swordsman takes. The menu of the game is a large map with various points of interest the gamer can move the swordsman to. Once at a point, the gamer can interact with various tribes and guards by selecting dialog options. The dialog options a nice way to help give the game a less rigid feel as the gamer can choose: to accept or reject a quest, demand more money, attack or defend a group, and many other options.

As the swordsman progresses through the game, he will earn more gold and more experience points. Using the gold, you are able to buy new weapons, armour, and other items to help your player increase his stats. Using the experience, you will level-up throughout the game. Each time you level-up, you will be shown 3 options of abilities that you can choose. Thankfully, this game does not follow the "freemium" mindset of many other games, which slowly forces you to pay for upgrade to beat the game. Instead, Reaper's free version is set up like a demo, where you are free to play until your character reaches level 10. The full unlocked game can be purchased in-game, after which there are no more addition in-game purchases necessary.

Combat makes up most of the game, and you will spend a lot of time fighting a variety of different characters in different



level locations. The game offers a great range of unique opponents that you will face. However, your best method to attack them stays relatively the same, which is to jump behind any land based troops and attack, then move away. With that in mind, there will be many other things attacking you as well as in level obstacles and enemy projectiles you will need to avoid.



Overall, Reaper is very simple to just pick up and play for a couple minutes or sit down for a while and enjoy. Everything about the game is very polished, and the artwork and graphics are terrific. I found all the levels and maps to be very captivating and interesting. Everything ran perfectly smooth on my ODROID-XU4, even though I have it underclocked with a passive cooler. Additionally, the game can be completely played using a Xbox 360 controller, or with a keyboard and mouse. I found the controller to be much easier to play with, and the game even showed all the getting started dialogs with the proper xbox controller buttons. I would strongly recommend any gamer looking for a fun action game to take a look at Reaper, as I know they will surely enjoy it.

GENTOO FOR THE ODROID-C2

CREATING YOUR OWN CUSTOM INSTALLATION

by @revOlt

gentoo linux

Gentoo Linux is a free operating system based on FreeBSD that can be automatically optimized and customized for just about any application or need. I have successfully gotten Gentoo's experimental stage 3 for ARM64 to boot and run on the ODROID-C2.

To begin, boot the C2 into any Linux distribution, making sure to backup important files just in case anything goes wrong. I used Ubuntu for the C2 since it is officially supported by Hardkernel.

Preparation

Prepare the Gentoo root filesystem by download the experimental Gentoo stage 3 for ARM64 from gentoo.org to a local directory in Ubuntu. In this example, we use /opt/gentoo as a local directory in Ubuntu and the latest stage 3 from date 3/24/2016, which you may change if you wish.

```
$ mkdir /opt/gentoo && cd /opt/
gentoo
$ wget http://distfiles.gentoo.
org/experimental/\
arm64/stage3-arm64-20160324.tar.
bz2
$ bzip2 -d stage3*
$ tar xvf stage3*
```

Then, copy over /lib/modules and /lib/firmware from Ubuntu to Gentoo. Gentoo stage 3 has /lib and /lib64. We also copy over the modules and firmware directories into Gentoo's /lib just in case:

```
$ cp -afv /lib/modules /opt/gen-
too/lib64/
$ cp -afv /lib/firmware /opt/gen-
too/lib64/
$ cp -afv /lib/modules /opt/gen-
too/lib/
$ cp -afv /lib/firmware /opt/gen-
too/lib/
```

Next, making sure that /opt/gentoo/etc/portage/make.conf is setup properly according to the Gentoo handbook available at http://bit.ly/1swpkQq. For the ODROID-C2, I leave the CFLAG untouched for now (CFLAGS="-O2 -pipe"), because adding in -march and -mtune seems to result in compile errors for some packages. I tried setting MAKEOPTS="-j5", but ran into low memory issue and the process was killed. However, using "-j3" instead resolves it. Just use conservative settings in the make.conf for now because it can always be fine-tuned later. The following commands will change root (chroot) into Gentoo:

```
$ cp -L /etc/resolv.conf /opt/gentoo/etc/
$ mount -t proc proc /opt/gentoo/proc
$ mount --rbind /sys /opt/gentoo/sys
$ mount --make-rslave /opt/gentoo/sys
$ mount --rbind /dev /opt/gentoo/dev
$ mount --make-rslave /opt/gentoo/dev
$ chroot /opt/gentoo /bin/bash
$ source /etc/profile
$ export PS1="(chroot) $PS1"
```

Configure Gentoo

```
$ passwd # set root password
$ useradd -m -G wheel -s /bin/bash yourname #add user youname
$ passwd yourname #set user yourname's password
$ emerge --webrsync
$ emerge --sync
$ echo "UTC" > /etc/timezone
$ emerge --config sys-libs/timezone-data
$ nano /etc/locale.gen # remove the relevant comment from the file
$ locale-gen

$ eselect locale list
$ eselect locale set X
#where X is the desired locale
$ env-update && source /etc/profile && \
export PS1="(chroot) $PS1"
```

Download the c2_init.sh script from http://bit.ly/2cWAjQP and place it as c2_init.start in the /etc/local.d directory.

```
$ chmod +x /etc/local.d/c2_init.start
$ rc-update add local default
```

We need to specify the Gentoo filesystem device in /etc/fstab. In this example, because I will be using partition 3 on my eMMC as the Gentoo root filesystem, I use the following command to update the file to match the following

```
$ nano /etc/fstab

# <fs>                  <mountpoint>
<type>                  <opts>
                        <dump/pass>

/dev/mmcblk0p3          /
ext4            errors=remount-
ro,noatime,nodiratime0 1
tmpfs           /
tmp             tmpfs
nodev,nosuid,mode=1777
      0 0
/dev/mmcblk0p1          /boot
vfat            defaults,rw,owner,fl
ush,umask=000           0 0
```

If you are booting using a microSD card, the device will also be identified as mmcblk0 after booting, so the above should still work, but you must change the partition number accordingly. For example, if you are booting Gentoo from a microSD card with Gentoo on partition 2, you would specify /dev/mmcblk0p2 as mount point / in your fstab (i.e., still use mmcblk0 but change p3 to p2).

Setup network

To configure the network drivers, type the following command:

```
$ emerge --noreplace net-misc/netifrc
```

To use DHCP on network interfaces eth0 and wlan0 (if you have a USB wifi device for example), edit /etc/conf.d/net as follows and save it:

```
config_eth0="dhcp"
config_wlan0="dhcp"
```

Then, type the following commands:

```
$ cd /etc/init.d
$ ln -s net.lo net.eth0
$ rc-update add net.eth0 default
$ ln -s net.lo net.wlan0
$ rc-update add net.wlan0 default
$ rc-update add sshd default
$ emerge net-misc/dhcpcd sys-kernel/linux-firmware ntp wpa_supplicant # ntp is needed as C2 has no persistent clock or else Gentoo would boot to 1970
$ rc-update add ntpd default
$ wpa_passphrase MYSSID myssidpassphrase > \
  /etc/wpa_supplicant/wpa_supplicant.conf # insert your MYSSID and myssidpassphrase accordingly
```

Note that to unmask some packages, you may want to use the command "emerge --autounmask-write" and then run "dispatch-conf" for ease of use. Then, exit chroot and unmount the chroot pseudo-filesystems:

```
$ exit
$ umount -l /opt/gentoo/dev{/shm,/pts,}
$ umount /opt/gentoo{/boot,/sys,/proc,}
```

Create the filesystem

Next, we need to create the filesystem on the partition for Gentoo, and then copy the Gentoo files prepared in the above steps over to that filesystem. In this example, I will be using partition 3 on my eMMC card for Gentoo:

```
$ mkfs.ext4 /dev/mmcblk0p3
$ mkdir /mnt/gentoo
$ mount /dev/mmcblk0p3 /mnt/gentoo
```

Then, we copy over the files from /opt/gentoo to /mnt/gentoo:

```
$ cp -afv /opt/gentoo/* /mnt/gentoo/
```

In this example, because the boot.

ini in Ubuntu's boot partition is pointing to the Ubuntu partition in /dev/mmcblk0p2 (mounted as /media/boot/ in Ubuntu), we must edit boot.ini to point it to the Gentoo root partition instead:

```
$ nano /media/boot/boot.ini
```

Change the root= variable to root=/dev/mmcblk0p3 in this line. For example:

```
setenv bootargs "root=/dev/
mmcblk0p3 rootwait ro ${condev}
no_console_suspend hdmimode=${m}
m_bpp=${m_bpp} vout=${vout}
fsck.repair=yes net.ifnames=0
elevator=noop
disablehpd=${hpd}"
```

Please make sure that the above line (i.e., the one that starts with the words setenv bootargs followed by the variables enclosed by the quotation marks) stays in one line

when you edit in nano. Web page formatting may likely cause the line to be displayed in multiple lines inadvertently.

Finally, reboot the device, and the C2 should boot into Gentoo, with the network eth0 and wlan0 working. The above method should also work if you are installing Gentoo and booting Gentoo on a microSD card instead of eMMC, as long as you adjust everything accordingly. For example, if you are installing Gentoo on a microSD card with partition 1 as boot and partition 2 as Gentoo root filesystem, then you would specify mmcblk0p2 as your / in Gentoo's filesystem in Gentoo's /etc/fstab before rebooting into Gentoo.

To prepare the microSD (/dev/sda1 for example), you would use sd_fusing.sh /dev/sda, and make sure the boot files (boot.ini, Image, meson64_odroidc2.dtb, uInitrd) from /media/boot/ in Ubuntu are copied over to /dev/sda1 before rebooting. You may want to keep your Ubuntu partition so that you can boot back into it and update kernel and /lib/modules /lib/firmware later. To do this, simply edit boot.ini in Gentoo's /boot and change the root= variable back to /dev/mmcblk0p2 (in my example) and reboot. Once kernel and /lib are updated in Ubuntu, you can copy them to Gentoo partition and reboot back into Gentoo by editing /media/boot/boot.ini as detailed above. One convenient way is to script this into both Gentoo and Ubuntu, and have them reboot back and forth periodically to update Ubuntu and copy the /lib/firmware and /lib/modules directories over to Gentoo, so that it is all done automatically in the background at night as you sleep.

In Gentoo, because we only started out with stage 3, once you have tuned your make.conf variables, you can then type the following command to recompile everything in world:

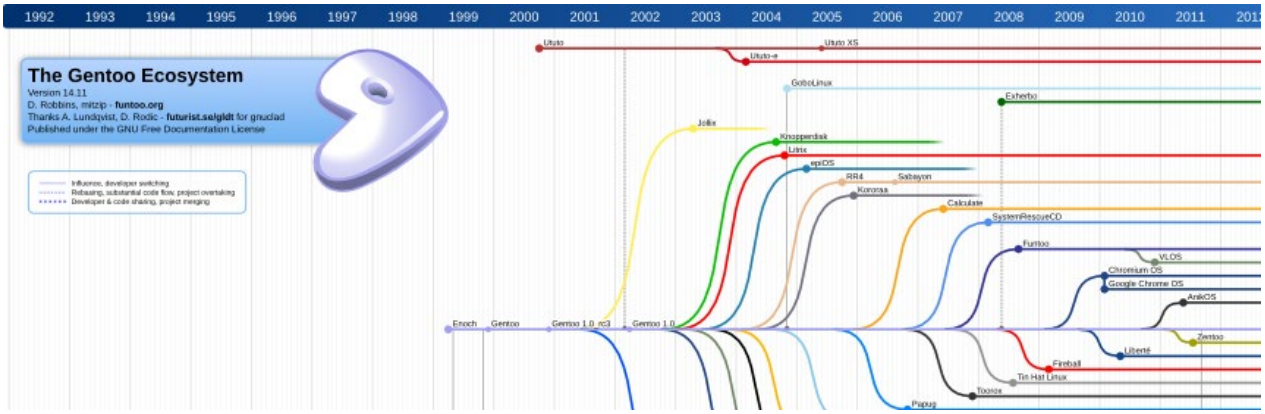
```
$ emerge -e world
```

Note that some packages may require kernel sources at /usr/src, so download or git clone the needed branch from http://bit.ly/2cISS9s into a directory in /usr/src named, for example, linux-c2-3.14, and then sym link (or eselect) it to the directory /usr/src/linux because emerge expects it there by default.

X Window compiles and runs quite nicely on C2 in Gentoo. If you wish to install it, you can use the following command:

```
$ USE="-llvm" emerge x11-base/
xorg-server
$ emerge openbox xterm # or
choose your favorite
$ nano /etc/X11/xinit/xinitrc #
remove the reference to twm and
replace it with exec openbox or
your favorite
$ startx
```

You may also want to consider active cooling solutions for the C2 if you compile software often. Looking at my log files, with the included stock CPU heatsink and without active cooling, the C2 runs a bit too hot for my personal liking when running big emerge jobs, topping out at around 70 degrees Celsius. You can find and keep track of the temperature during big compilation jobs by searching for temp in /sys and looking at the relevant file. It may be a good idea to keep it cool with a fan when you compile or at least keep it at some safe distance away from the back of the TV. For questions, comments and suggestions, please visit the original forum post at http://bit.ly/2ctWugX or the Gentoo main page at http://www.gentoo.org.



MEET AN ODROIDIAN

SEBASTIEN CHEVALIER (@PTITSEB), GLSHIM DEVELOPER

edited by Rob Roy

Our man, Sebastien in his office

Please tell us a little about yourself.

My name is Sebastien Chevalier. I'm 43, French, and married with two children.

How did you get started with computers?

When I was around 10 years old, my father came back home with a computer. It was an obscure French computer called Hector, which plugged into the TV and came with 3 cassettes. Since it was the early 1980s, it didn't have disks, and ran Space Invaders, a useless mortgage software, and BASIC. After spending many hours playing Space Invaders, I started to look at the BASIC compiler. There were also two printed manuals about BASIC, so I learned the language in order to make more games.

What attracted you to the ODROID platform?

My first contact with ODROID devices was a couple of years ago. After porting FreeSpace and FreeSpace 2 to OpenPandora, @meveric contacted me to help him with porting it to the ODROID platform. I was happy that my port could be used on other device other than initially intended. Later, my collaboration with @meveric intensified, and I started adding more and more ODROID code to my ports.

How do you use your ODROIDs?

My ODROID is plugged to my main TV. I use it for casual browsing, playing a few games, testing of new ports, and as a media player.

Your continuing work on the GLShim package is very popular, especially among game enthusiasts. What motivated you to work on the project?

When I started porting games, I did the OpenGL to Open GLES porting by hand. I made a few games, and updated some major engines, such as FreeSpace, Jedi Knights II and III. I then followed the GLShim project created by @lunixbochs (`http://bit.ly/2d4PHek`). At first, I wasn't completely convinced, since my manual ports were good, and sometimes even better than a simple GLShim port. Still, some games were very complicated to port, like Armagetron Advanced and Scorched Earth 3D, so I tried the GLShim approach. I then forked @linuxboch's GitHub project in order to have my own sources, and worked in

Sebastien's first computer, the Hector

Sebastien visited Como Lake in Italy last summer

a "port by port" approach by taking a game or piece of software and making it work. Some games and software applications use functions that are difficult to implement, or crash and don't render correctly, so I analyze what is happening, then add a function to fix it. Other times I just give up, move on to other software, and come back later after GLShim has evolved.

These days, GLShim is working pretty nicely, with mostly all of the OpenGL 1.5 functions implemented, and many GLX functions too. I can now run games like Minecraft, idTec 1, 2 or 3, the Serious Engine 1 games, and software like Blender.

Which ODROID is your favorite and why?

I currently only own an ODROID-XU4, which is a nice machine. It's powerful on both the CPU and GPU side, but the active cooling makes it less than ideal for my current use as an HTPC. I'll probably try the passively-cooled C2, which seem powerful as well, especially now that the 3D driver is available.

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

I would like to see some kits based on DIY designs from previous issues of ODROID Magazine.

What hobbies and interests do you have apart from computers?

I don't have many hobbies. I play games, but that is still computer-related. I like to travel, especially on holidays with my family (see the picture of beautiful Como Lake in Italy where I was this summer). I spend most of my free time on evenings and weekends on computer. I'm a programmer in my job too, so I guess I'm easily spending 12 hours a day in front of a computer screen.

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

Learning programming can take time. My main advice is to take on projects, but not overly ambitious ones. To learn, you can start with some simple games. For example, if you have never programmed before, take a game like "Find a number", where the computer chooses a number between 1 and 999 which you have to guess, and have the computer tell you if you are above or below. Then, try a simple text-based Tic-Tac-Toe game, then a basic Break Out game, which should be a first step into graphics. The language that you use is not that important, so try BASIC, Pascal, C, C++, Python, or Lua. What is important is that the objective is achievable, so you learn by doing things and don't get discouraged. You must allow yourself time to become a good programer and shouldn't compare yourself too early with other people.

Sebastien enjoying a meal