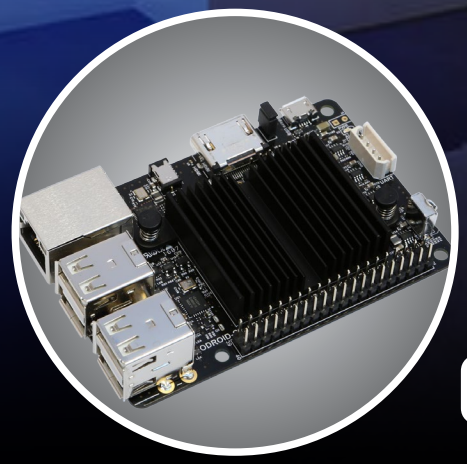
Android Gaming • Advanced Imaging Sensors • Android Wifi Stack

ODROID Magazine

Year Three
Issue #35
Nov 2016



Ultra-HD 4K ODROID *Ambilight*

Create a synchronized video light display

- Linux Gaming: Get Serious with the Serious-Engine
- Android Nougat: OpenJDK-based Java and a new graphics API

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-C2 and XU4 devices to EU countries! Come and visit our online store to shop!

Address: Max-Pollin-Straße 1
85104 Pförring Germany

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

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

EDITORIAL

One of the most versatile peripherals for the ODROID is the Arduino, which can be programmed as a standalone controller for many projects, from robots to home automation. A simple project to get started with the Arduino is to create an Ambilight system, which is a stunning background light display that synchronizes itself with live video. The engineers at Hardkernel demonstrated it at ARM TechCon 2016, and wrote a guide for you to easily create the same stunning light show in your own home.

To further enhance your viewing experience, we present a tutorial on setting up a MythTV front end as well as an article on enabling accelerated video playback in an ODROID-C2 web browser. For more experienced DIY enthusiasts, Miltiadis presents his lights controller with SMS notifier project that can be adapted and expanded to any IoT application, and Jörg shows us how to set up an alarm system with window sensors. Andy expands upon our previous Docker series with up-to-date information, Tobias introduces us to the Serious gaming engine, Nanik describes the Android WiFi stack, and Bruno has fun with Ancestor, a visual stunning Android game with amazing gameplay.

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

Bruno lately is fiddling with 2 of his ODROIDs, playing games and being amazed by the responsiveness of his machines with this new and amazing system. He is making sure that he never runs out of gaming column ideas for the readers that discover new games along with him!

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.

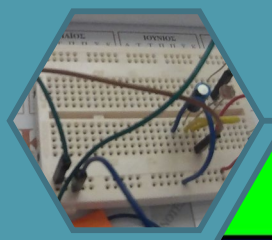
Venkat Bommakanti, Assistant Editor

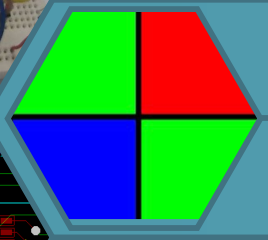
I'm a computer enthusiast from the San Francisco Bay Area in California. I try to incorporate many of my interests into single board computer projects, such as hardware tinkering, metal and woodworking, reusing salvaged materials, software development, and creating audiophile music recordings. I enjoy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

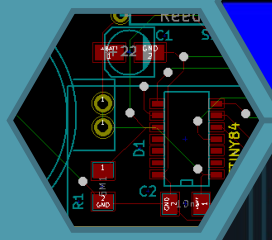
Josh Sherman, Assistant Editor

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







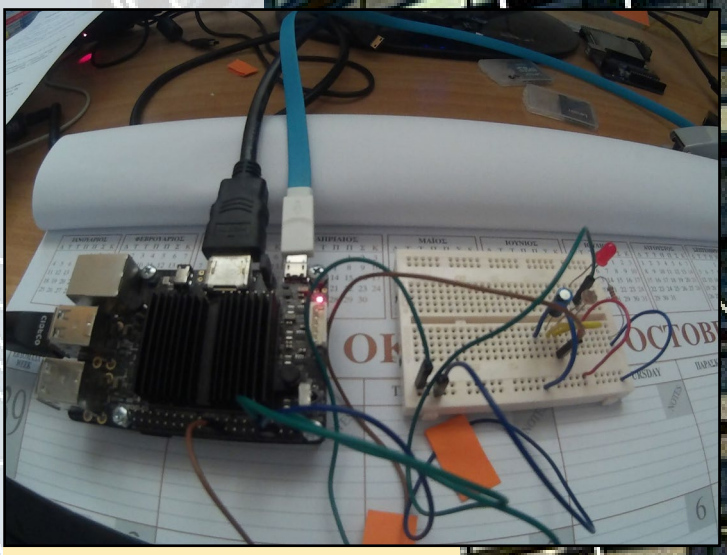
BUILDING AN IOT DEVICE USING AN ODROID-C2

STREET AND HOME LIGHTS CONTROLLER WITH SMS NOTIFIER

by Miltiadis Melissas

It is common knowledge that cities consume a lot of energy operating their street-lighting infrastructure. Individual users face similar situations for controlling the lighting in their homes efficiently and effectively. The IoT lighting solution presented in this article, which is based on Hardkernel's ODROID-C2, an excellent 64-bit quad-core single board computer (SBC) (`http://bit.ly/2bWxgrK`), can help create a safe, energy-efficient environment with smart capabilites. Smart street-lighting and home lighting sensors can easily be connected to the network as Internet of Things (Iot).

The sensor, which is a photoresistor in this project, can turn lights on and off upon successful reading in order to ensure the lowest energy consumption and proper operation. Moreover, home users can be notified from an IoT device by means of SMS messages sent to mobile phones. The SMS messages can notify the users of the exact timing the lighting is set to on/off back in their home and reporting possible malfunctions.



The assembled lighting solution using an ODROID-C2 and C Tinkering Kit

This is the third project in my series of tutorials regarding Internet of Things (IoT) using an ODROID-C2. This is also the first time we make use of a photoresistor/photocell sensor. Our previous projects were built and operated using only actuators, such as LEDs and servos. This article will guide you on how to drive such an electronic component, controlling it as an input, by using the WiringPi library inside Python programming language, and thus setting the basis for our next IoT project: Wine Preserver and Notifier.

The IoT device works under the normal light conditions during typical daily exposure, and the photoresistor keeps the LED off under these circumstances. However, when it get dark, the photoresistor triggers the ODROID-C2 and the LED turns on and blinks, simulating the operation of the street/home lights at night. The interesting thing is that when this happens, the ODROID-C2 notifies the user that this operation has started successfully by sending an SMS message to his/her mobile phone or tablet. This is a complete IoT device that makes use of a sensor (photoresistor), an actuator (LED) and a cloud service (SMS messaging).

Building the circuit

We will use a breadboard in order to avoid any soldering and the hassle of designing a PCB. We will connect various circuit components with the ODROID-C2 GPIO pins using Dupont Jumper Wires, as shown in this page.

Hardware

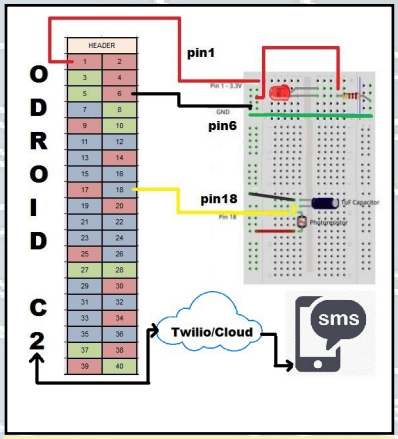
ODROID-C2 running Ubuntu
USB power supply 5V/2A and cable, use the right one provided by Hardkernel's store (http://bit.ly/1X0bgdt)
Breadboard and Dupont Jumpers (male to female)
1 X Photocell/Photoresistor
1 X 1uF Capacitor
1 X LED
1 X 220 Ohm Resistor

Software

Ubuntu 16.04 v2.0 available from Hardkernel at (http://bit.ly/2cBibbk)
Python language for programming. Fortunately for us Ubuntu 16.04 v2.0 from Hardkernel comes pre-installed with this programming tool
WiringPi Library for controlling ODROID-C2 GPIO pins. For instructions on how to install this go to Hardkernel's excellent setup guide available at (http://bit.ly/2ba6h8o)
Python language for programming the IoT device

Building our IoT device

As mentioned previously, we will use a breadboard to build our IoT device with the electronic components and Dupont jumper wires. It's a good idea to disconnect the power supply from the ODROID-C2 before connecting anything on its pins, because you can destroy it with a short circuit if you make a wrong connection accidentally. Double check with the schematic in this article, and make the correct connections before you power it up.



Circuit diagram

For connections, we used the male to female Dupont wires. The female side of this kind of jumper connects to the male header of the ODROID-C2 and the other one -male- connects to the holes of the Breadboard. Please refer to Hardkernel's pin layout schematic at next page as you create the connections, which is also available at http://bit.ly/2aXAlmt:

Physical Pin1 provides the VCC (3.3V) to our circuit, and we connect it on the second vertical line of our Breadboard.

Since we are going to use Pin6 as the common Ground, we connect that to the second vertical line of our Breadboard, near the edge.

The photoresistor/photocell is connected to physical pin18 on one of its side, the other one goes to VCC (3.3V). Please note that this red Dupont wire/jumper connected to the vertical line of our Breadboard. Kindly refer again to our schematic in Figure 2 for the correct connections. Extra care must be given to the polarity of the capacitor (1uF), since we need to connect its negative side marked by (-) symbol with the common Ground. The positive side of the capacitor is connected to the photoresistor through the yellow Dupont wire and from there to physical pin18. We will explain the role of

capacitor (1uF) in the next paragraph.

Finally, the LED is connected to physical pin7 for its anode (+) while the cathode (-) is connected of course to the common Ground.

That's it! All of our physical wiring is now connected.

The role of the resistor and capacitor

For this circuit, we need to use the 3.3v out from the ODROID-C2 Pin 1, as well as Ground (GND) of course. We connected these from the ODROID-C2 to the Breadboard. The operational LED is connected to pin7 through a 220 Ohm resistor in order to limit the amount of current that flows through the LED. The presence of the resistor ensures that the LED components will be keeping safe under an accidental very large current.

The role of capacitor is different, however. This is because we need the capacitor to act like a bucket and the photoresistor like a thin pipe. To fill a bucket up with a very thin pipe takes enough time that you can figure out how wide the pipe is by timing how long it takes to fill the bucket up halfway. In this case, our bucket is a 1uF capacitor. So, the photo resistor is connected through the 1µF capacitor to pin18 of the ODROID-C2 and the negative side of the capacitor is connected to the common Ground. Since the hardware setup is now done, let us see how we can send SMS message from our IoT device.

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 Pin Layout

Using Twilio

Twilio is a Python package that sends text messaging (SMS). Twilio is not part of the standard Python library, but it's one of the thousands of external Python packages that are available for us to download and use.

Python developers usually use one of the two common utilities to automatically download and setup necessary folders and files: "easy-install" and "pip". "easy-install" comes with the setuptools Python library, which is standard for Python and pip comes with the "pip" library. "easy_install" and "pip" are executed in the terminal that can be used to install Python packages. Since the ODROID-C2 Linux image (http://bit.ly/2cBibbk) comes pre-installed with Python, it is very easy to install Twilio on our IoT device. The only step we need to take is starting up the Terminal application and type in:

```
$ sudo easy_install twilio
```

Enter the administrator password to give easy-install permission to write to our system folders, which is "odroid" on the official Hardkernel Linux image. Alternatively, if you want to install Twilio with pip, which is the installer for Python, you have to first install pip on ODROID-C2:

```
$ sudo easy_install pip
$ sudo pip install twilio
```

If you want to check whether Twilio was properly installed, enter the Python command to enter the Python interpreter in the Command Prompt or Terminal application and type in these 2 commands:

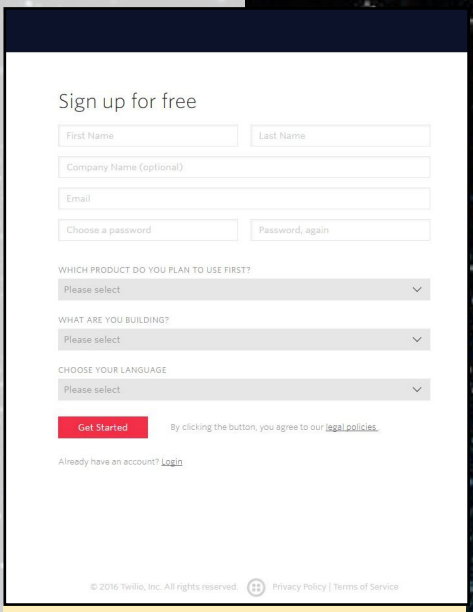
```
> import twilio
> print(twilio._version_)
```

If it prints a version number of Twilio, the setup has been completed properly.

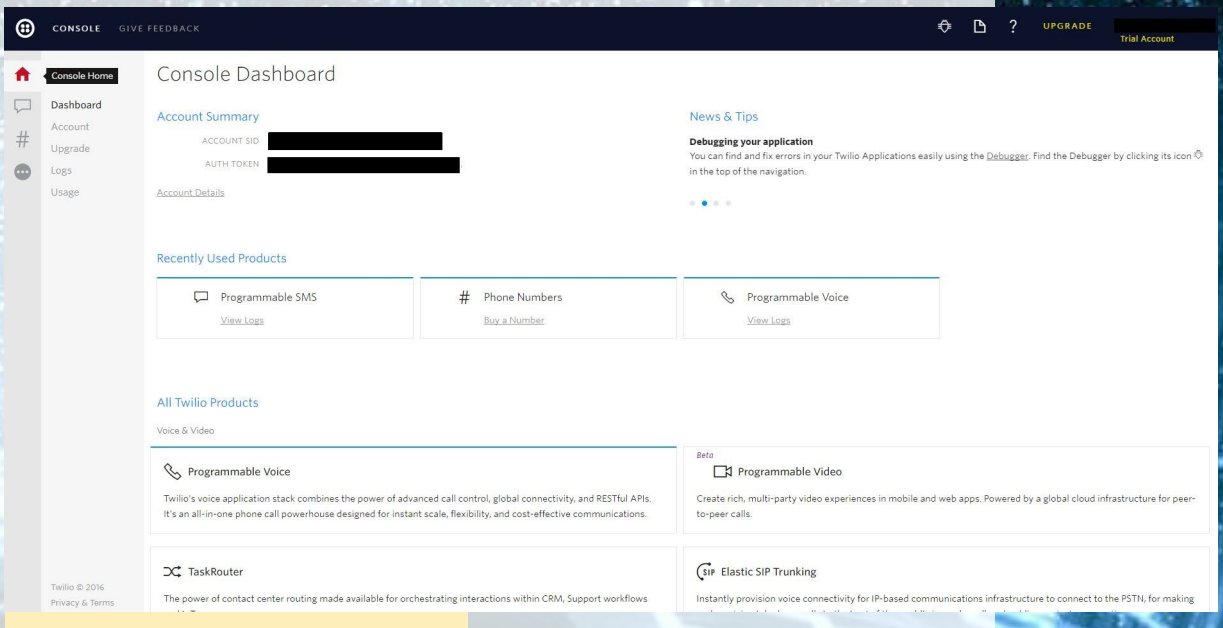
Twilio registration

Go to the Twilio registration page at `http://bit.ly/296QVCl`, and sign up for free, as show in Figure 4. Twilio needs your mobile number, so provide it in the appropriate field. Twilio will then send you a verification code on the phone that you have previously registered in order to verify that you are not a software bot. Enter the code in appropriate the box, and Twilio will give you a phone number. Make a note of the phone number, and continue with the registration process.

Finally, you will land on a page with lots of activities, including make a call, send an SMS message, receive a call, and receive an SMS message. From this page, we need Twilio's API authorization token. Look for the button titled "Go to your account" and click it. On the dashboard page is the account SID and the authorization token, as shown in Figure 5. Next, you have to copy and paste it to our program. Note that your screen may look a bit different if it is your first time you are logging to Twilio. In my case, my account SID authorization token were at the top of the page (blanked out). Yours may be somewhere at the bottom.



Twilio signup web page



Twilio dashboard with API token

Explaining Twilio code

You can find the sample code below from the Twilio Python Helper Library available at http://bit.ly/2dyGB8n. All of the source code relevant to the Twilio service is available from the GitHub repository at `http://bit.ly/2dndZ0s`.

```
from twilio.rest import TwilioRestClient
# Your Account SID from www.twilio.com/console
account_sid = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
# Your Auth Token from www.twilio.com/console
auth_token  = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"

client = TwilioRestClient(account_sid, auth_token)

message = client.messages.create(body="Hello from ODROID-C2",
  to="+306972438526",    # Replace with your phone number
  from_="+12052364839")  # Replace with your Twilio number

print(message.sid)
```

You will notice that inside Twilio, there is a folder called "rest", and inside that folder is a class called "twiliorestclient". We make use of that class in the following code snippet:

```
<from Twilio.rest import Twiliorestclient>
```

In the following line of code, this line of code, we assign a variable client to twilioRestClient for verification:

```
<client=twilioRestClient (account_sid, auth_token)>
```

Finally, with the following line, we create the message and print it or actually sending it to our mobile phone:

```
<message=client.sms.messages.create>
```

Sending the SMS message

First, copy and paste the account SID and auth-token to your program. Next, change the body of the text message to something like: "Hello from ODROID-C2" as I did in the below example. In the field called "to", change it to your phone's mobile number. In the field called "from_", you have to fill in your Twilio number: this is the number Twilio gave you upon registration. If you have not written it down, go back to your Twilio account (http://bit.ly/2dpPpM1) and on the top of the page find the numbers tab and click it. You will get your phone numbers from Twilio, as shown in Figure 6. Now save and run the program and see if it works:

```
from twilio.rest import TwilioRestClient
# Your Account SID from www.twilio.com/console
account_sid = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
# Your Auth Token from www.twilio.com/console
```

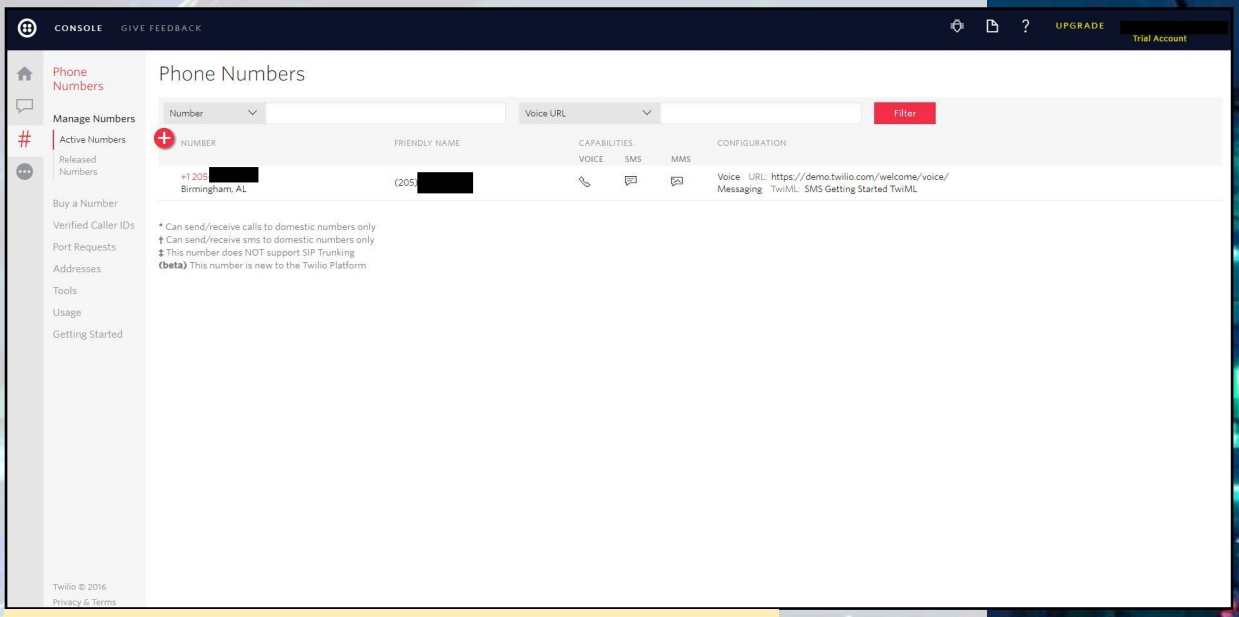
```
auth_token  = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"

client = TwilioRestClient(account_sid, auth_token)

message = client.messages.create(body="Hello from ODROID-C2",
    to="+30XXXXXXXXXX",       # Replace with your phone number
    from_="+120XXXXXXXXX")    # Replace with your Twilio number

print(message.sid)
```

Your phone should notify you that you have gotten an SMS message. The final step is to connect this SMS code with our photoresistor and make our IoT device a smart one.



Twilio setup screen showing the phone number used in the code snippet

Connecting Twilio to the photoresistor

Now that we know how Twilio is working, let's see how to connect it with our photoresistor code. The tricky part is how to calibrate the photoresistor according to our light conditions in the room. There is always a threshold that we need to find out by some trial and error in order to trigger the IoT device, such as the blinking of the LED and the sending of the SMS message to the user at the same time. Remember that the blinking of the LED simulates the normal operation of lights during the night, in the street or at home, and that the SMS sent to user confirms normal operation. Please study the code below and then follow along as I explain line by line what is happening.

```
#!/usr/bin/env python

# Example for RC timing reading for ODROID-C2
# Must be used with wiringpi2

import wiringpi2 as odroid, time
from twilio.rest import TwilioRestClient
```

```
DEBUG = 1
odroid.wiringPiSetup()

LEDpin = 7
odroid.pinMode(LEDpin,1)

def RCtime(RCpin):
        reading = 0
        odroid.pinMode(RCpin,1)
        odroid.digitalWrite(RCpin,0)
        time.sleep(0.1)

        odroid.pinMode(RCpin,0)
        # This takes about 1 millisecond per loop cycle
        while (odroid.digitalRead(RCpin) == 0):
                reading += 1
        return reading

def Send_SMS():
      account_sid = "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"  # Your Account
SID from www.twilio.com/console
      auth_token  = "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"   # Your Auth Token
from www.twilio.com/console

      client = TwilioRestClient(account_sid, auth_token)

      message = client.messages.create(body="Hello from Python",
              to="+30XXXXXXXXXX",      # Replace with your phone number
              from_="+120XXXXXXXXX")   # Replace with your Twilio number

      print(message.sid)

while True:

     print RCtime(5)     # Read RC timing using physical pin #18
     time.sleep(300)
     if  (RCtime(5)>2500):
       Send_SMS()
     for i in range (0,300):
       if (RCtime(5)>5500):
           odroid.digitalWrite(LEDpin,1)
           time.sleep(0.02)
           odroid.digitalWrite(LEDpin,0)
           time.sleep(0.02)
```

First, we import the wiringpi2 module and we create the object "odroid" because we want to control the GPIO pins of our ODROID-C2:

```
<import wiringpi2 as odroid>
```

There is a detailed tutorial from Hardkernel's excellent support page on how to

download and install WiringPi to your ODROID-C2 running Ubuntu at http://bit.ly/2ba6h8o. Next, we import TwilioRestClient from twilio.rest, something that we have explained in detail in the previous paragraph:

```
<from twilio.rest import TwilioRestClient>
```

Then, with the following line of code, we reference the GPIO wiring according to the table provided by Hardkernel for ODROID-C2, as shown in Figure 2:

```
<odroid.wiringPiSetup()>
```

We assign the pin7 for the LED:

```
<LEDpin=7>
```

Immediately after that, we set it as an output

```
<odroid.pinMode(LEDpin,1)>
```

In the next section of code, we define the function called RCtime. This is a very important function for measuring up the levels of light in the room. We keep track of these levels with a counter:

```
<reading = 0>
```

Next, we setup the relevant pin i.e pin5 (physical pin18) as an output:

```
<odroid.pinMode(RCpin,1)>
```

We then write to that pin:

```
<odroid.digitalWrite(RCpin,0)>
```

We alternate state almost immediately after that using time.sleep(0.1):

```
<odroid.pinMode(RCpin,0)>
```

Pin5 is set now to low, and on the next line of code we read its state, adding +1 to our counter:

```
<while (odroid.digitalRead(RCpin) == 0):>
        <reading += 1>
```

Finally, we check if the level of darkness is under its threshold:

```
 <RCtime(5)>2500>
```

The number 2500 is calculated with some trial and error endeavours looking for the right threshold according to our light conditions in the room. If the light conditions are above this limit, we call the function Send_SMS and we sent the SMS message via Twilio at the same time that we make the LED blink. Please

note that the LED blinks for an interval equal to the time that we set for our IoT device to check for the right light conditions in the room. In this example, that interval is 300 seconds, or every 5 minutes. Of course you can set your own intervals for checking.

```
<time.sleep(300)=for i in range (0,300)>
```

Those time intervals must be the same to ensure the right timing. In order to make the LED blink, the pin is set to high and then low with a short time interval of 0.02 milliseconds:

```
for i in range (0,300):
…..
<odroid.digitalWrite(LEDpin,1>
<time.sleep(0.02>
<odroid.digitalWrite(LEDpin,0>
<time.sleep(0.02>
```

Bringing it all together

Now it's time to run and test our code. Copy and paste the entire code to your Python IDLE document (Integrated Development and Learning Environment) under the name OdroidSMS.py. Remember that all Python scripts have the extension *.py. You can start IDLE from your Ubuntu desktop by simply clicking the Applications Menu (Applications -> Programming -> IDLE). After the file has been saved, run it with sudo privileges from a command prompt after navigating to the directory where the file resides:

```
$ sudo python OdroidSMS.py
```

This is the basic idea of a smart device, and if I can do it, you can do it too. What will be your next step? Take this guide, study it carefully and then expand upon it by creating another, even a more sophisticated IoT device with your ODROID-C2.

HOW TO GET FANCY COLOR IMAGES FROM A SIMPLE SENSOR IMAGE

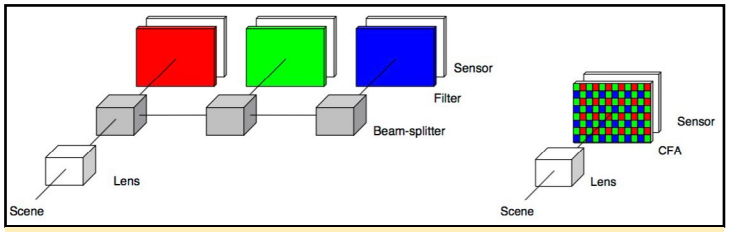
USING THE BAYER PATTERN TO CREATE AN RGB COLOR IMAGE

By withrobot@withrobot.com

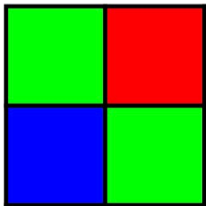
It is well known that the color of a pixel can be represented by a mixture of three primary colors: red, green, and blue. For this, many people might think that a single pixel in a camera's sensor also has three colors: red, green, and blue. For example, in a 1024 x 1024 image, it is generally assumed that there is the same amount of pixels, 1024 x 1024, of red, green and blue colors. However from a manufacturing point of view, it is very complicated and expensive to put three different types of color sensors in one location. Therefore, a beam splitter is usually used to light up the sensors on different sensor panels. As a result, this approach is prohibitively complex, bulky and expensive.

A more practical and feasible alternative is to have monochrome sensors with an accompanying color filter. Here, the filter has the same number of cells as the image pixels. For example, in a 1024 x 1024 image, we use 1024 x 1024 monochrome sensors with a color filter of 1024 x 1024 cells with three colors: red, green and blue. Figure 1 shows two diagrams the the leftmost is of multi-sensors with beam-splitting. The rightmost diagram is of monochrome sensors with color filter array, or CFA.

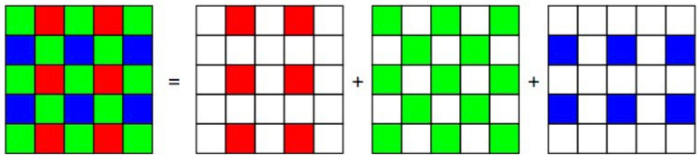
Although various patterns can be used for a CFA, the Bayer pattern is the most common. In the next column we show the basic 2 x 2 form of the Bayer pattern which has two greens, one red, and one blue filter. We use more green subpixels to mimic the sensitivity of human eyes which are more tuned to detect intensities of the color green. The Bayer pattern can be thought of the combination of red, green and blue patterns as shown below.



Two different structures of color sensor: multi-color sensors(left) and single monochrome sensors(right)



Basic form of the Bayer pattern: The Bayer pattern is the combination of red, green and blue patterns.

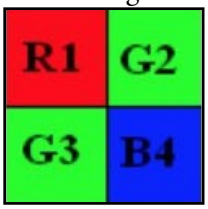


For example, the first row of a 1024 x 1024 image is made up of 512 red pixels and 512 green pixels. Similarly, the second row is made up of 512 green pixels and 512 blue pixels. Therefore, if we use one byte of data for each pixel, the total data size of 1MB of a 1024 x 1024 image is composed of 0.25MB red data, 0.25MB of blue, and 0.5MB of green data. This is a great size reduction for image data. The data size of an image from a setup with three different color sensor panels would be 3MB or three times the size compared the 1MB image from the Bayer pattern.

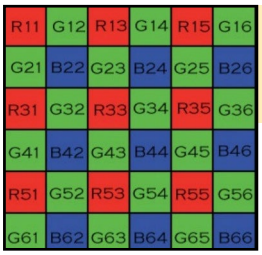
However, we inevitably lose detailed color information by using the Bayer pattern color sensor. For example, if we look at the top left pixel in figure 3, we only get the intensity of green. Therefore, we have to "guess" the other color values for this pixel. Generally, the interpolation is used to estimate the missing values. One of the simplest methods is the Pixel Doubling Interpolation. Using the nomenclature used in Figure 4, we get the full RGB color intensities for each pixel using the following formula:

Top left pixel: (R, G, B) = (RI, G2, B4)
Top right pixel: (R, G, B) = (RI, G2, B4)
Bottom left pixel: (R, G, B) = (RI, G3, B4)
Bottom right pixel: (R, G, B) = (RI, G3, B4)

Although we get the full color data with the least amount of calculation using this technique, we also get the worst quality image. To enhance the image quality, more pixels in the neighborhood of the pixel being filled in are used, in addition to using a more complicated formula. One example of this is the Bilinear Interpolation method.



2 x 2 Bayer pattern block for the Pixel Double Interpolation.



6 x 6 Bayer pattern block for the Bilinear Interpolation.

Pixel R33: (R, G, B) =
(R33, (G23+G34+G32+G43)/4, (B22+B24+B42+B44)/4)

Pixel G34: (R, G, B) =
((R33+R35)/2, G34, (B24+B44)/2)

Pixel G43: (R, G, B) =
((R33+R53)/2, G43, (B42+B44)/2)

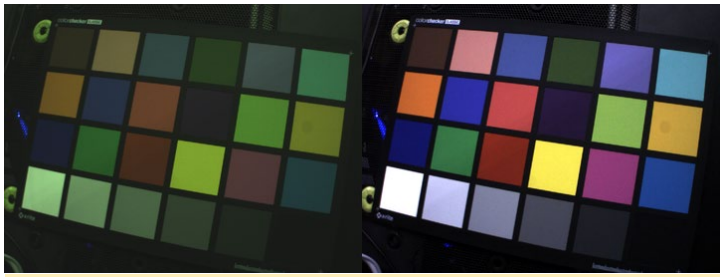
Pixel B44: (R, G, B) =
((R33+R35+R53+R55)/4, (G34+G43+G45+G54)/4, B44)

For more information about the various interpolation techniques, you can refer to "Image Demosaicing: A Systematic Survey by Xin Li, Bahadir Gunturk, and Lei Zhang (http://bit.ly/2eHnGGm).

The problem with Bilinear Interpolation is the poor color quality. To overcome this limitation, many CMOS sensor manufacturers use a special processor known as an Image Signal Processor, or ISP. This further enhances the image obtained by interpolating the Bayer pattern image.

Although the usefulness of a global shutter camera is well known, as discussed in the August 2016 ODROID Magazine article titled "Understanding oCam's Global Shutter" (http://bit.ly/2ee4sJ9, many ODROID users requested a global shutter camera after the release of the oCam-1MGN-U, the monochrome global shutter camera. However, no global shutter color camera has been provided to be used with ODROIDs because there is no color sensor with both a global shutter and ISP functionality.

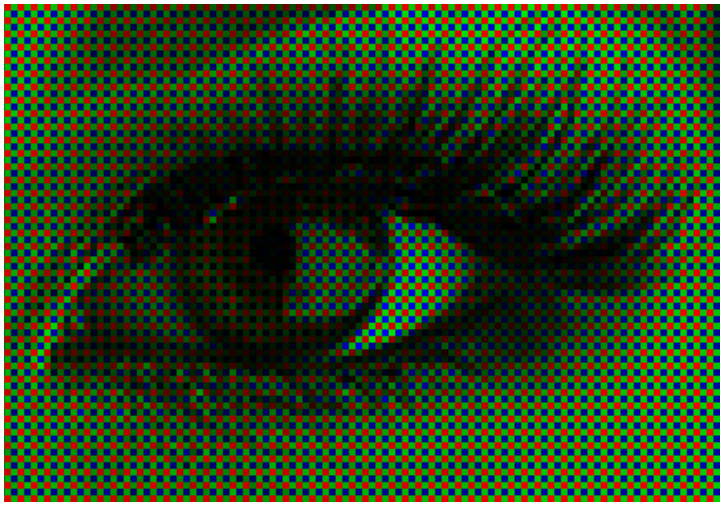
A lot of effort has been given to solve this problem through the use of software, instead of waiting for appropriate sensor hardware to become available. Fortunately, a new type of global shutter color camera has been developed for ODROIDs using a proprietary algorithm. This new camera, the oCam-1CGN-U, will be available around December 2016. Figure 6 shows the striking improvement in the color quality.



Original color image obtained by using Bilinear Interpolation of Bayer image (left) and the color image enhanced through a proprietary enhancement algorithm (right) applied to the original image.

The new camera will have the following specifications:
Sensor: OnSemi AR0134 Bayer Color CMOS image sensor
Lens: Standard M12 lens (changeable)
Image sensor size: 1/3 inch
Image resolution: 1280 x 960
Shutter: Electric global shutter
Interface: USB 3.0 super-speed

In next month's article, an interesting example will be developed using this new global color shutter camera with the ODROID platform.



ALARM CENTRAL

PART I - RF24 WINDOW SENSOR AND MIRF LIBRARY

by Jörg Wolff

This is the first part of my series about my Alarm Central project that uses an ODROID-C1 running Android. The project consists of the Alarm Central Android app, ultra low power window sensors, and ultra low power motion sensors, which are not yet ready. The sensors communicate with the ODROID-C1 using Nordic Semiconductor nRF24L01 2.4Ghz modules. This article details the ultra low power window sensors and the communication library, mirf, which I ported to Android.

During my first tests, I decided to use the ODROID-C1 instead the ODROID-C2, because the latter does not have a native SPI interface and the bitbang SPI driver for the ODROID-C2 is too slow to use with the nRF24L01 modules. The development of other parts the system, such as the door lock and fingerprint sensor, is ongoing. The Alarm Central project is not yet installed in my house but, as soon as I finish the case for the ODROID-VU7+ and the ODROID-C1, I will install it. In case of an alarm, the app will send a short message through the internet to a smartphone.

Alarm Central Home

Window sensor

The window sensors are based on an ATTiny84 processor. I designed a small 24mm x 60mm board which contains a reed contact, a connector for the nRF24L01, an ISP connector for flashing the processor, a holder for a CR2450, and some additional parts. The PCB was ordered from Itead Studio, and the components were hand soldered, which took about 20 to 30 minutes per board.

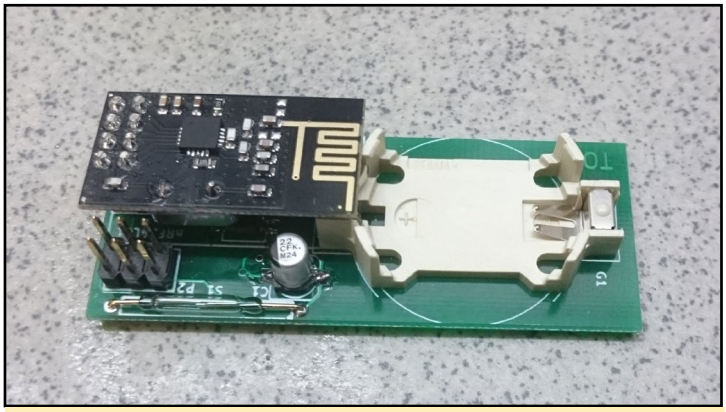
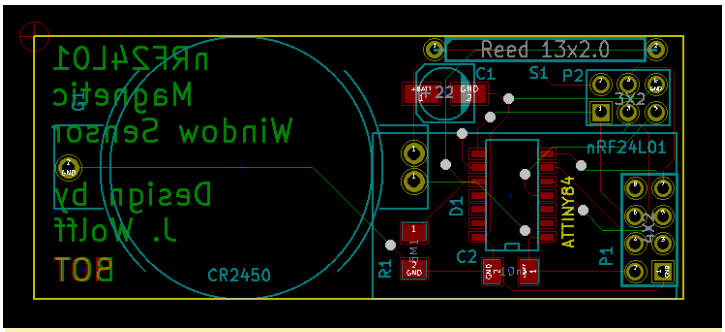


Figure 2 Nrf24 Window Sensor



Window Sensor pcb

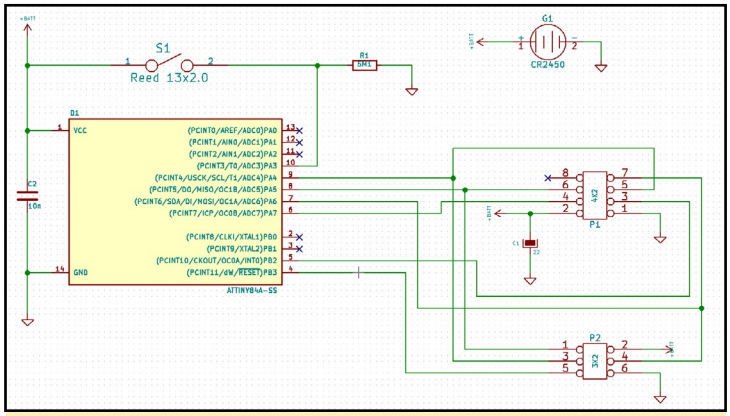
Partlist

Printed board Attiny 84A-SSU SO-14 NRF24L01 module
Battery Holder HU2450 Renata
Reed Contact NO 13x2.0
Resistor 5MI SMD1206
Capacitor 22u/16V 4.3x4.3
Capacitor 10n/50V 3.2x1.6
Pin Strip 2x3 2.54
Female strip 2x4 2.54
Neodym Magnet 10x1

(Dimensions are in mm)

It would be possible to design the board to be smaller, but the larger size accommodates a 650mAh CR2450 battery that provides a long battery life. The ATtiny is designed to sleep for 4 seconds, then wake up and send a 20 bytes message to the Alarm Center. If the reed contact changes state, the ATtiny wakes up and sends a message to the ODROID-C1. During sleep mode, the total current consumption of all components is about 6µA. When the ATtiny is awake, the current jumps for a short period to around a couple of mA. The overall average total current is about 17µA. With a battery capacity of 650mAh this give us a battery life of 3 to 4 years. Without message encryption, the average current would be even less and the battery life would be about 5 years. To reach this low current in sleep mode, the Brown Out Detection is disabled, this makes it impossible to store data in the EEPROM. Occasionally, due to power cycling, there is some data loss such as node number or the AES key. This made me implement data storage in the flash. With data being stored in flash, there is no longer any data loss when power cycling.

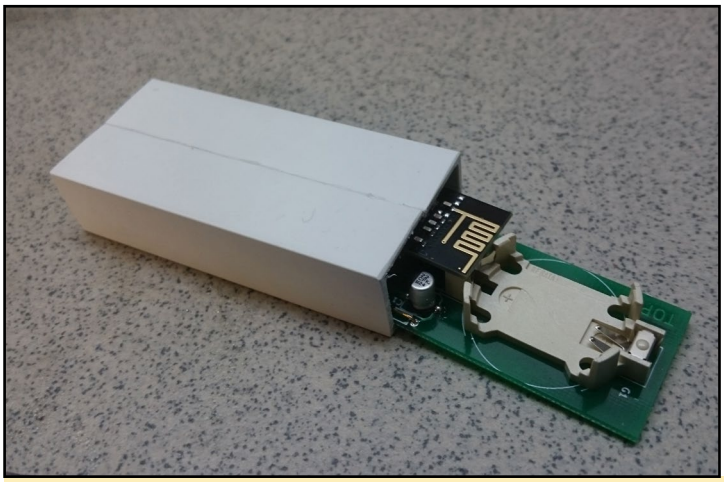
On the sensor board's first boot, it sends its data unencrypted with the node number, 255. The Alarm Central receives this message and does auto node numbering and returns the AES key. This only happens when Alarm Central is offline and the user has made authentication. For a short time, communication is open. The code for the sensor can be found on Github at http://bit.ly/2dHQkrS.



Circuit diagram

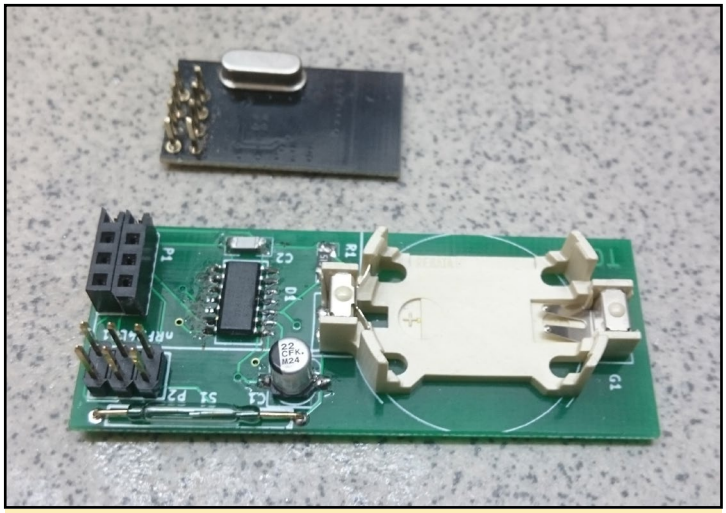
I could not find a small plastic case that fit my sensor. So, I used a plastic U-profile 15 mm x 15 mm (9/16" x 9/16") strip, and cut two 68mm (2 5/8") pieces and glued them together.

The most complicated part seemed to be soldering all the small SMD parts, but with the right technique and a little practice, it goes smoothly. It's best to begin with the ATtiny and only solder one pin so you can adjust the position a bit, then the other pins. It works great if you solder a few of the pins together, since a solder bridge can be removed with desoldering braid and a little soldering flux. You should not forget to clean any excess soldering flux off the components with acetone or a universal cleaner.



Sensor and Case

To reduce the height of the PCB and components, the quartz can be desoldered from top and soldered to the bottom of the nRF24L01 board. Also, the female 2x4 strip can be wetted down 1 or 2 mm (1/16") and the pins of the 2x4 strip can be cut about 2 mm (1/16"). The total height should be about 13 mm (½") to make them fit into the case. The KiCad project can be found at http://bit.ly/2eviBAu.



Sensor and nRF24L01

Mirf Library

The mirf library is responsible for wireless communication, and was ported to Android. Basically, it is the same code that is used on the ATtiny and the ODROID-C1. The two key differences in the code are with the SPI interface code and a C++ wrapper on the ODROID code. To build the library of ODROID, first install the Android NDK. Next, build the library from jni folder by running the following command:

```
$ ../../ndk-build -B
```

You can find the source code on Github at http://bit.ly/2eiANjl. To use this library in an Android app, it needs a wrapper library such as this:

```
/*
    Model of Mirf wrapping library ported to ODROID-C1 / Android

    Copyright (C) <2016>  <Jörg Wolff>

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.
    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.
    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

#include <jni.h>
#include <stdio.h>
#include <stdlib.h>
#include <android/log.h>
#include <mirf.h>

#ifdef __cplusplus
extern "C" {
#endif

#define LOG_TAG "com.jw.mirf"

#define LOG_D(...) __android_log_print(ANDROID_LOG_DEBUG, LOG_TAG, __VA_ARGS__)
#define LOG_F(fn_name) __android_log_write(ANDROID_LOG_DEBUG, LOG_TAG, "Called : " fn_name )

static JavaVM *java_vm;
mirf* receiver;

jint JNI_OnLoad(JavaVM* vm, void* reserved)
{
    JNIEnv* env;
    if (vm->GetEnv(reinterpret_cast<void**>(&env), JNI_VERSION_1_6) != JNI_OK) {
        return -1;
    }

    // Get jclass with env->FindClass.
    // Register methods with env->RegisterNatives.

    system("insmod /system/lib/modules/spicc.ko");
    system("insmod /system/lib/modules/spidev.ko");

    return JNI_VERSION_1_6;
}

//mirf(uint8_t _cepin, uint32_t _freq, uint8_t _spi_channel, uint8_t _payload_size, uint8_t _mirf_CH);
JNIEXPORT void JNICALL
Java_path_to_your_app_MirfSetup(JNIEnv * env, jobject obj, uint8_t ce, uint32_t speed, uint8_t spi_channel, uint8_t size, uint8_t mirf_channel) {
    receiver = new mirf(ce, speed, spi_channel, size, mirf_channel);
    //LOG_D("Setup");
}

//void config(void);
JNIEXPORT void JNICALL
Java_path_to_your_app_MirfConfig(JNIEnv* env, jobject obj) {
    if (receiver != NULL) receiver->config();
}

//void reconfig_rx(void);
JNIEXPORT void JNICALL
```

```
Java_path_to_your_app_MirfReConfigRx(JNIEnv* env, job-
ject obj) {
    if (receiver != NULL) receiver->reconfig_rx();
}


//void reconfig_tx(void);
JNIEXPORT void JNICALL
Java_path_to_your_app_MirfReConfigTx(JNIEnv* env, job-
ject obj) {
    if (receiver != NULL) receiver->reconfig_tx();
}


//void set_address(uint8_t pos, uint8_t* address);
JNIEXPORT void JNICALL
Java_path_to_your_app_MirfSetAddress(JNIEnv* env,
jobject obj, jbyte pos, jstring address) {
    if (receiver != NULL){
        const char *nativeString = env-
>GetStringUTFChars(address, 0);
        receiver->set_address(pos, (uint8_t*)na-
tiveString);
        LOG_D("SetAddress: %s", nativeString);
        env->ReleaseStringUTFChars(address, na-
tiveString);
    }
}


//uint8_t receive_data(void* buf);
JNIEXPORT jbyteArray JNICALL
Java_path_to_your_app_MirfReceiveData(JNIEnv* env,
jobject obj, jbyte size) {
    if (receiver != NULL) {
        jbyte *data=(jbyte *)
malloc(size*sizeof(jbyte));
        receiver->receive_data(data);
        jbyteArray result=env->NewByteArray(size);
        env->SetByteArrayRegion(result, 0, size,
data);
        delete[] data;
        return result;
    }
    return 0;
}


//uint8_t transmit_data(void* buf);
JNIEXPORT void JNICALL
Java_path_to_your_app_MirfTransmitData(JNIEnv* env,
jobject obj, jbyteArray array) {
    if (receiver != NULL) {
        jbyte *buf = env->GetByteArrayElements(array,
NULL);
        receiver->transmit_data(buf);
        env->ReleaseByteArrayElements(array, buf, 0);
    }
}


//uint8_t status(void);
//uint8_t max_rt_reached(void);


//uint8_t data_ready(void);
JNIEXPORT int JNICALL
Java_path_to_your_app_MirfDataReady(JNIEnv* env, job-
ject obj) {
    if (receiver != NULL) return receiver->data_
ready();
    LOG_D("MirfDataReady:return 0");
    return 0;
}


//uint8_t read_register(uint8_t reg, uint8_t* buf,
uint8_t len);
//uint8_t read_register(uint8_t reg);
//uint8_t write_register(uint8_t reg, const uint8_t*
buf, uint8_t len);
//uint8_t write_register(uint8_t reg, uint8_t value);
//void config_register(uint8_t reg, uint8_t value);
//uint8_t get_data(void* buf);
//uint8_t send_data(void* buf);


//void power_up_rx(void);
JNIEXPORT void JNICALL
Java_path_to_your_app_MirfPowerUpRx(JNIEnv* env, job-
ject obj) {
    if (receiver != NULL) receiver->power_up_rx();
}


//void power_up_tx(void);
JNIEXPORT void JNICALL
Java_path_to_your_app_MirfPowerUpTx(JNIEnv* env, job-
ject obj) {
    if (receiver != NULL) receiver->power_up_tx();
}
```

```
//void power_down(void);


//uint8_t flush_rx(void);
JNIEXPORT int JNICALL
Java_path_to_your_app_MirfFlushRx(JNIEnv* env, job-
ject obj) {
    if (receiver != NULL) return receiver->flush_rx();
    return 0;
}


//uint8_t flush_tx(void);
JNIEXPORT int JNICALL
Java_path_to_your_app_MirfFlushTx(JNIEnv* env, job-
ject obj) {
    if (receiver != NULL) return receiver->flush_tx();
    return 0;
}


//void start_listening(void);
JNIEXPORT void JNICALL
Java_path_to_your_app_MirfStartListening(JNIEnv* env,
jobject obj) {
    if (receiver != NULL) receiver->start_listen-
ing();
}


//void stop_listening(void);
JNIEXPORT void JNICALL
Java_path_to_your_app_MirfStopListening(JNIEnv* env,
jobject obj) {
    if (receiver != NULL) receiver->stop_listening();
}


//void delay_us(unsigned int howLong);
JNIEXPORT void JNICALL
Java_path_to_your_app_MirfDelayMicroSeconds(JNIEnv*
env, jobject obj, int us) {
    if (receiver != NULL) receiver->delay_us(us);
}


#ifdef __cplusplus
}
#endif
```

The Mirf library and the functions need some glue code, so the Java app can call the C++ library:

```
static {
    System.loadLibrary("mirf_android");
}


public native int MirfSetup( byte ce, int speed, byte
spi_channel, byte size, byte mirf_channel);
public native void MirfConfig();
public native void MirfReConfigTx();
public native void MirfReConfigRx();
public native void MirfPowerUpRx();
public native void MirfPowerUpTx();
public native void MirfSetAddress(byte pos, String
address);
public native byte[] MirfReceiveData(int size);
public native void MirfTransmitData(byte[] data);
public native int MirfDataReady();
public native void MirfStartListening();
public native void MirfStopListening();
public native int MirfFlushRx();
public native void MirfDelayMicroSeconds(int us);
```

And to create a mirf object, this code as example in the onCreate() function:

```
/*
* Some needed constants for the mirf object
*/
byte pin_ce = 6; //Header pin 22
byte spi_channel = 0;
byte length_payload = 20;
byte mirf_channel = 5;
int spi_speed = 4000000;
/*
* Setup the mirf communication
*/
MirfSetup(pin_ce, spi_speed, spi_channel, length_pay-
load, mirf_channel);
MirfConfig();
```

In a loop,or a HandlerThread, the messages can be read from the sensors, as shown in the following code snippet:

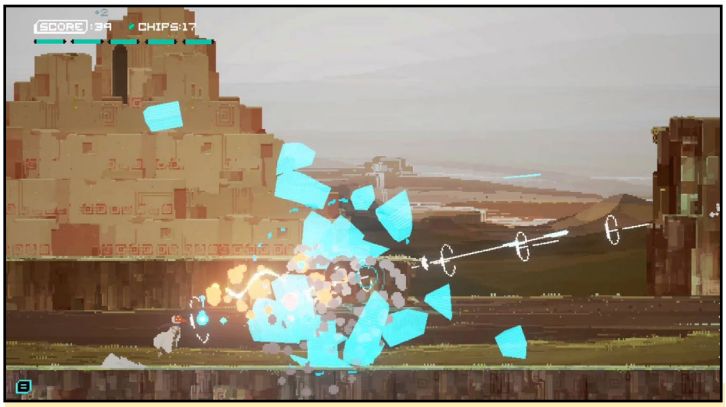
ANCESTOR

A GAME FULL OF FUN WITH PERFECT GAMEPLAY, VISUALS AND DETAILS

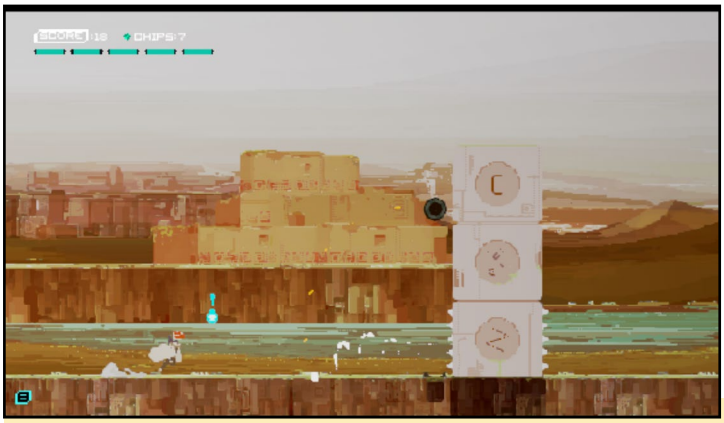
by Bruno Doiche

While we and our beloved ODROIDS are still far from emulating the Playstation 3, where we can enjoy playing Journey to the point of exhaustion, we can get a game with similar visuals, puzzles and the added bonus of being an endless runner game with bosses! Ancestor was written by a brother and sister production team that absolutely loved Mass Effect. They teamed up with their father, who is a programmer, and the rest is up to you to figure out how far you can go in this absolutely enjoyable game. Grab your joystick and think fast!

https://play.google.com/store/apps/details?id=com.supermegaquest.ancestor



The look is similar to Journey (PS3), but the gameplay is frantic



In Ancestor, you solve puzzles to progress and have fun


```
while (true) {
    MirfPowerUpRx();
    MirfFlushRx();
    MirfStartListening();
    while (MirfDataReady() == 0) {
        MirfDelayMicroSeconds(250);
    }
    MirfStopListening();


    inbuffer = Arrays.copyOf(MirfReceiveData(length_
payload), 16);


    //Do something with inbuffer.


    try {
        Thread.sleep(5L);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
```

The wiring of the nRF24L01 module to the ODROID-C1 is as follow:

CI.Header.19 – nRF24L01.6	**(MOSI)**
CI.Header.21 – nRF24L01.7	**(MISO)**
CI.Header.23 – nRF24L01.5	**(SCK)**
CI.Header.22 – nRF24L01.3	**(CE)**
CI.Header.24 – nRF24L01.4	**(CSN)**
CI.Header.1 – nRF24L01.2	**(VCC)**
CI.Header.6 – nRF24L01.1	**(GND)**

In the next part of this series, I will share more information on the RF24 motion sensor, the Alarm Central App itself, and a nice handmade case for the ODROID-VU7+.

ULTRA-HD 4K AMBILIGHT

CREATE A SPECTACULAR SYNCHRONIZED VISUAL BACKGROUND FOR YOUR HOME THEATER

by Charles Park and Brian Kim

Ambilight, short for "ambient lighting", is a lighting system for television developed by Philips in which lighting effects are created around the TV that corresponds to the video content. You can achieve a similar effect by using a strip of RGB LEDs and software that samples the image on the screen, then colors each individual LED in a different shade accordingly. In this article, we shall see how to use an ODROID to achieve this.

Figure I - Ambilight on ODROID-C2

Requirements

We used an Arduino for controlling the LEDs and are using an ODROID-C2 in order to run Kodi media player as the interface for the video content. The hardware components are listed below.

- ODROID-C2
- Arduino UNO
- USB cable Type A – Type B
- 5V/6A power Supply
- 32GB eMMC Module C2 Linux
- WS2801 LEDs
- DC Plug Cable Assembly 2.5mm
- 4-pin connector cable x 3

We used an Arduino to control LEDs and ODROID-C2 to do everything else, such as playing the media file, capturing the video frame, and sending the LED color data format via USB serial interface. Arduino receive the color data format from ODROID-C2, and then sets the color of each LED.

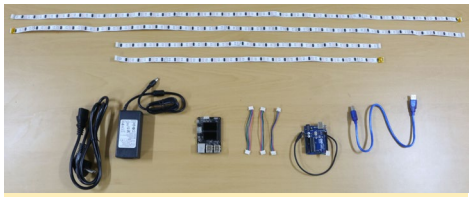


Figure 2 - Hardware components for Ambilight

Hardware setup

The first step is to set up the wiring on the Arduino board. There are two wiring areas: the 4-pin LED connector, and the ODROID-C2 DC plug cable. The 4-pin LED connector connects to the WS2801 LEDs which are able to be controlled via the SPI interface.

4-pin LED connector cable

- Red: VCC
- Black: Ground
- Blue: SCK (13)
- Green: MOSI (11)

DC Plug Cable

- Red: VCC
- Black: Ground

The WS2801 is a constant current LED driver, and is designed for indoor/outdoor LED displays and decorative LED lighting system. In order to mount LEDs into the TV, we cut the LEDs roll with an alternative size from the TV height and width size. Each cut LEDs need to be connected with 4-pin connector cables.

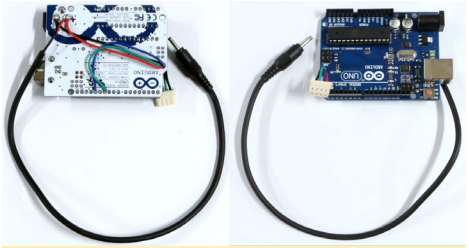


Figure 3 - Arduino Wiring

Software setup

There are three kinds of main software for DIY ambilight: Arduino LED control firmware, Hyperion, and Kodi media player. The Arduino is connected to the ODROID-C2 via USB serial device (ttyACM0). We can easily develop the Arduino firmware in Linux natively on the ODROID-C2 using the Arduino IDE:

Figure 4 - LED Wiring

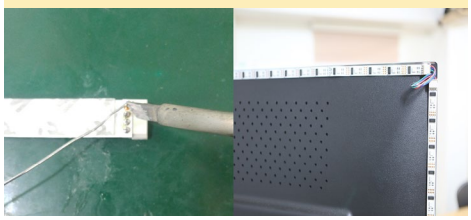


Figure 5 - Ambilight mounted on the TV

```
$ sudo apt-get update
$ sudo apt-get install arduino
$ cd /usr/share/arduino/librar-
ies/
$ sudo git clone \
  https://github.com/adafruit/
Adafruit_NeoPixel.git
```

The Adafruit NeoPixel library is for controlling LEDs. The LED control firmware source code is available for download at https://git.io/vPVqT:

```
$ wget https://git.io/vPVqT -O
odlight.ino
$ arduino
(Ctrl + O) -> (Select /home/
odroid/odlight.ino file)
(Ctrl + R) Verify / Compile
(Ctrl + U) Upload
```

Of course, the Arduino needs to be connected to ODROID-C2 during the firmware upload. You can get more information about Arduino IDE software on the Arduino home page at http://bit.ly/212hc7p.

To describe Ambilight behavior, background software captures the video frame while the media file is displaying, and then sends the RGB data to the LED control device. Hyperion is an open source Ambilight implementations that runs on many platforms, which uses the video capture driver of the ODROID-C2 for getting the video frame data. However, the video capture driver allocates 8 megabytes DMA area, so ODROID-C2 Linux kernel needs more contiguous memory:

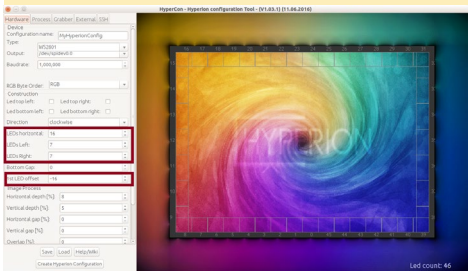
```
$ sudo apt-get update
$ sudo apt-get install git build-
essential
$ git clone --depth 1 \
  https://github.com/hardkernel/
linux.git -b odroidc2-3.14.y
$ cd linux
$ make odroidc2_defconfig
$ make menuconfig
Device Drivers --->
  Amlogic Device Drivers --->
    Video Decoders --->
      [*] Amlogic Video Capture
support
  Generic Driver Options --->
    *** Default contiguous memory
area size: ***
      (12) Size in Mega Bytes
$ make -j4
$ sudo make modules_install
$ sudo mv /media/boot/Image /
media/boot/Image.back
$ sudo mv /media/boot/ /media/
boot/Image.back
$ sudo mv /media/boot/meson64_
odroidc2.dtb \
  /media/boot/meson64_odroidc2.
dtb.back
$ sudo cp arch/arm64/boot/Image /
media/boot/
$ sudo cp arch/arm64/boot/dts/me-
son64_odroidc2.dtb \
  /media/boot/
$ sudo sync
$ sudo reboot
```

Hyperion is a good choice for LED color control software because it requires less processing power, works quickly and effectively, and also provides an easy configuration. Furthermore, Hyperion is compatible with the Amlogic platform on the ODROID-C2, even though it does not officially support it yet. However, it is not complicated to add support for the ODROID-C2 in Hyperion:

```
$ sudo apt-get update
$ sudo apt-get install cmake
libqt4-dev \
  libusb-1.0-0-dev python-dev
libxrender-dev \
  python libasound2-dev zlib1g-
dev
$ git clone --depth 1 \
  https://github.com/mdrjr/c2_
aml_libs.git
$ cd c2_aml_libs
$ sudo make
$ sudo make install
$ cd
$ git clone --depth 1 \
  --recursive https://github.com/
bkrepo/hyperion.git
$ cd hyperion
$ mkdir build
$ cd build
$ cmake -DENABLE_DISPMANX=OFF \
  -DENABLE_SPIDEV=OFF -DENABLE_
AMLOGIC=ON \
  -DCMAKE_BUILD_TYPE=Release
-Wno-dev ..
$ make -j4
$ sudo make install
```

Hyperion needs a configuration file, which can easily be generated by the Hypercon configuration program, which is available at http://bit.ly/2dRqkgO. Even if Hypercon cannot generate a complete configuration file for ODROID-C2, the program is useful for LED position setting. There are three options in order to set LED positions: LEDs horizontal, LEDs Left and LEDs Right. The first LED offset option is for adjusting LED starting point, and we can also set the direction to clockwise or counterclockwise. To get the JSON format configuration file after finished the LED position setting, just click the Create Hyperion Configuration button.

Figure 6 - Hypercon



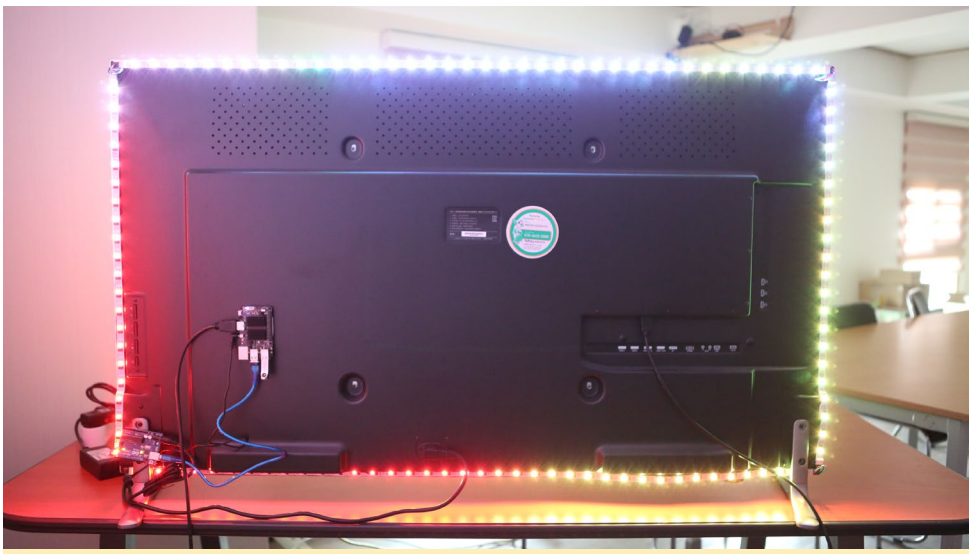


Figure 7 - Ambilight Running

The "leds" option in the generated JSON configuration file via Hypercon needs to be copied to the default configuration file (https://git.io/vPrKR). All of the other options in the default JSON configuration file can remain set to the default values:

```
$ wget https://git.io/vPovU -O /
etc/hyperion/hyperion.config.json
```

Open the generated configuration file via Hypercon, then overwrite the "leds" option from the generated configuration file, and write it to the file /etc/hyperion/hyperion.config.json.

Playing a movie

To play Ambilight on the ODROID-C2, run the Hyperion daemon in the background, then play the movie in Kodi:

```
$ hyperiond /etc/hyperion/hyper-
ion.config.json &
$ kodi
```

ODROID-C2 also supports 4K H265 movies, which requires setting the double_write_mode option, as detailed at http://bit.ly/2doJtDU:

```
$ echo 1 | sudo tee \
  /sys/module/amvdec_h265/param-
eters/double_write_mode
```

For more information, please refer to the Adalight Project http://bit.ly/1EnG6zZ), our previous Ambilight article in ODROID Magazine (http://bit.ly/2dOPWsk) and our ODROID forum threads at http://bit.ly/2eyPrUr and http://bit.ly/2eo1DrY.

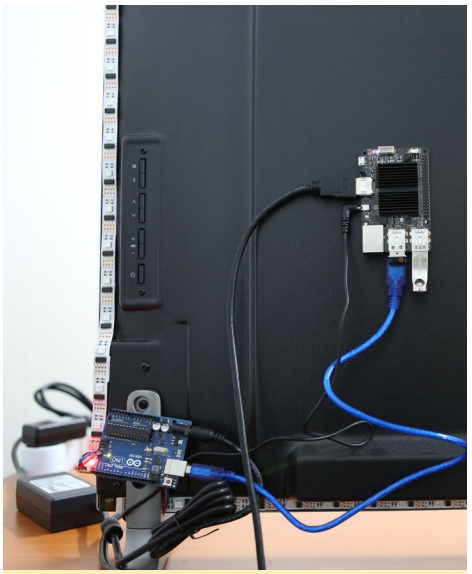


Figure 8 - Ambilight closeup

DOCKER 101

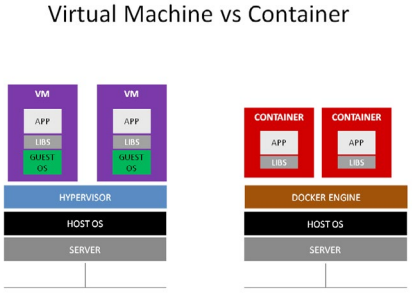
PART I - WHY DOCKER?

by Andy Yuen

The article "Why People user Docker" (http://bit.ly/2f5nRyi), sums up Docker nicely. Docker interests me because it allows simple environment isolation and repeatability. I can create a run-time environment once, package it up, then run it again on any other machine. Furthermore, everything that runs in that environment is isolated from the underlying host (much like a virtual machine). And best of all, everything is fast and simple.

Docker uses a virtualization technology called containers. Containers use the Linux kernel feature "cgroups" to isolate between containers and other processes running on the host, and "namespaces" to make available a set of system resources and present them to a process as if they are dedicated to that process. Container technology is different from virtual machine technology in that it does not need a hypervisor nor a guest operating system. Containers only require an application packaged with dependent binaries and libraries in an image. For virtual machines, the virtualization is all the way down to the device level, while that for containers is down to the operating system level only.

Figure 1 - Virtual Machine versus Container



Compared to virtual machines, containers are smaller and consume less resources while operating with improved performance. Since your application is isolated in a Docker image together with its dependencies (binaries and libraries), you can build it once and run it on any Docker host with the same computer architecture the image was built on. This means that Docker images built for Intel x86 architecture will not run on ARM64 machines and vice-versa.

There are many more reasons for enterprises to use including Docker CI/CD, DevOps, blue/green deployment, rolling updates, etc. I just don't have space in this article to cover them all. An interesting fact about Docker is that anytime you use Google, such as searching, Gmail, or Google Docs, you are being issued a new container.

Using Docker

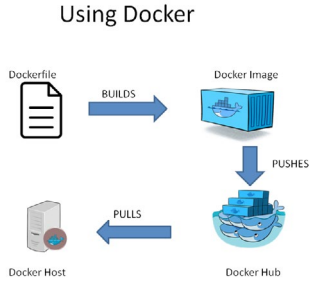
Figure 2 shows the flow of working with Docker. The list below details the flow:

1. Someone creates a Dockerfile
2. The Dockerfile is used to build an image, which is your application with its dependent binaries and libraries.
3. The image is pushed to Docker Hub, which serves as a central repository for the Docker images. You can find a large number of Docker images that you can download and use if they suit your purpose.
4. A user pulls an image from Docker Hub and runs the image in a container. Multiple replicas of the application can be run on the same Docker host or on a Docker cluster on demand.

Overview

In this tutorial, I will cover all of the activities listed above, and is split into 2 parts. The first part covers the classic Docker commands to start and stop containers and let them communicate with each other. The second part cov-

Figure 2 - Using Docker



ers Docker swarm mode, which is new in Docker version 1.12. Swarm mode is all about orchestration, which means clustering and scheduling of where to run the containers, and how many replicas should be started in a cluster.

ODROID Magazine published several Docker articles back in early 2015, so what has changed since then? For one thing, kernels that support Docker were not that common in stock Operating Systems for ODROID. In order to run Docker, one has to tinker with kernel builds which, for most people, myself included, is too much trouble. However, kernel features that are required for Docker have been incorporated in many operating systems. Another thing to note is that Docker has been designed and built for 64-bit machines. Back in 2015, all ODROIDs were based on 32-bit architecture only. The ODROID-C2 is the first 64-bit ODROID to date. Although Docker can be, and has been, adapted to run on 32-bit ARM architecture, this is the first 64-bit implementation. Let's explore its capabilities in this tutorial.

Prerequisites

In order to follow this tutorial, you must have the following in place:

1. ODROID-C2 running an OS with kernel features required by Docker enabled

I use Armbian Xenial server, which is based on Ubuntu. How I decided on using the Armbian Xenial server is documented in my blog at http://bit.ly/2dyTUGr. I had a look at the Armbian website at http://bit.ly/2exEWPH recently, and noticed that only the Jessie server, which is based on Debian, is available for download.

2. ODROID-C2 with Docker engine installed

To install the Docker engine on your machine, issue the following commands:

```
$ docker run -d -p 3306:3306 \
 --name mysql \
 -e MYSQL_USER=fishuser \
 -e MYSQL_PASSWORD=fish456 \
 -e MYSQL_DATABASE=fish \
 -v /media/sata/fish-mysql:/u01/
my3306/data \
 mrdreambot/arm64-mysql
```

If you are using a different OS from mine, you have to use your OS's package management command such as yum, dnf, or apt-get. If Docker is not available in your OS's repository, try the Docker v1.12.1 binaries that I used along with an installation script at http://bit.ly/2ejY1FE. These binaries have been tested on Armbian Xenial and Jessie servers. For Part 1 of the tutorial, either docker.io v1.10, v1.11 or v1.12 will work fine. Part 2 will require v1.12.

In order to avoid having to add "sudo" in front of every Docker command you issue during the tutorial, you should add your user name (login) to the "docker" group as follows, then reboot the system and then log back in:

```
$ docker run -d \
 -p 8080:8080 --name fish \
 -e MYSQL_SERVER=192.168.1.100 \
 -e MYSQL_PORT=3306 \
 mrdreambot/arm64-fish
```

3. ODROID-C2 with a working Internet connection

Your ODROID-C2 will need an internet connection to pull down images from the Docker Hub during this tutorial.

Basic Docker commands

In the tutorial that follows, for simple commands whose output is self-explanatory, I shall just include the screenshots for the command and output. For more complicated commands, I shall also explain the command syntax and what the options mean. If you want help for the options available to a certain command, just enter the command and append --help as shown in Figure 6.

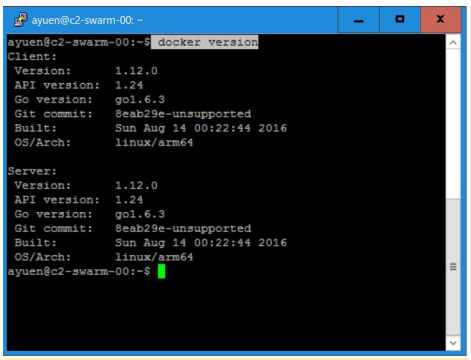


Figure 3 - Checking the Docker version

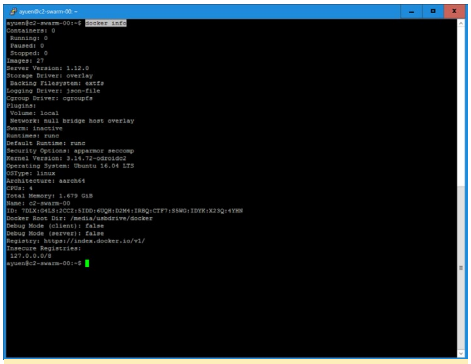


Figure 4 - Finding out more about Docker

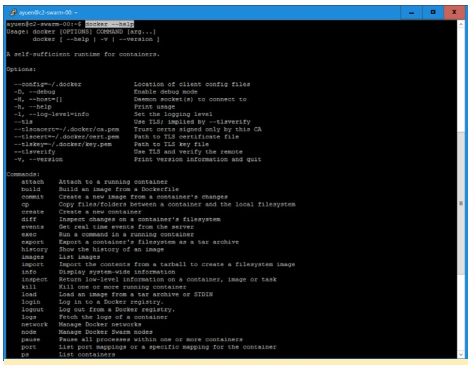


Figure 5 - Getting Help

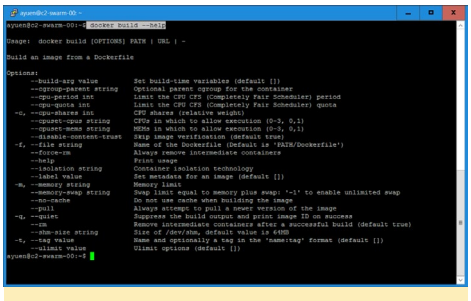


Figure 6 - Docker Build Help

Running your first container

Make sure you have an Internet connection, and issue the following command:

```
$ docker run -d -p 80:80 --name
```

```
httpd \
  mrdreambot/arm64-busybox-httpd
```

Options:

-d means run in the daemon mode

-p 80:80 means map port 80 of the container to the host's port 80 so that you can access the application on your ODROID-C2's port 80

--name gives a name to the container you just started. This is optional, although I highly recommend you to always give it a name so that you can refer to your container easily. You can identify running containers even without using the --name option by issuing the "docker ps" command, which is described later. Since the image "mrdreambot/arm64-busybox-httpd" is not already on your Docker host, Docker will download it from the Docker hub.

Navigate your web browser on your PC to your Docker host, which is your ODROID-C2 machine. You should get the page shown in Figure 7, which is our Docker "Hello World" program.

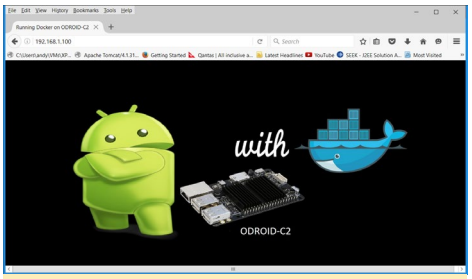


Figure 7 - ODROID Docker test page

Connecting to your running container

You can connect to your running container by issuing the following command:

```
$ docker exec -it httpd /bin/sh
```

The options are as follows:

-it means Interactive tty mode

httpd is the name we gave the container when we started it

/bin/sh is the command we want to run when we connect to the container.

In this case, we want to start the command shell

We can run any command available to the command shell. In this example, we run the ping command. When we are done, we just type "exit" to exit the container. The container is still running after you exited from the interactive session.

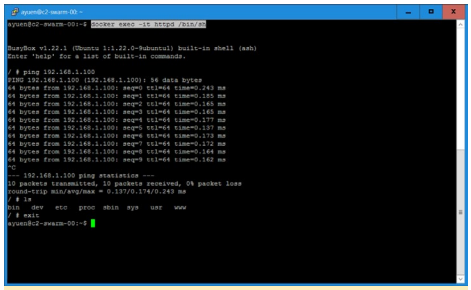


Figure 8 - Docker Exec

We could have started the container in the interactive mode when we started the container. For example, instead of using the following command:

```
$ docker run -d -p 80:80 --name
httpd \
  mrdreambot/arm64-busybox-httpd
```

We could have used this command:

```
$ docker run -it -rm \
  mrdreambot/arm64-busybox-httpd
bin/sh
```

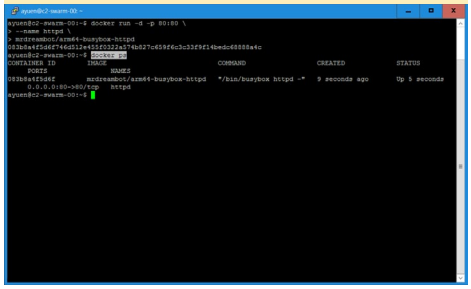
The options used are:

-it means interactive tty mode

-rm means remove the container when we exit from the shell

/bin/sh means replace the default entry point (starting the httpd server) with the /bin/sh command.

Figure 9 - Docker PS command output



Stopping and removing your active running containers

To list the running containers issue the following command:

```
$ docker ps
```

To stop and remove the httpd container we started, issue the following commands:

```
$ docker stop httpd
$ docker rm httpd
```

Sometimes when you run a container in an interaction session and exit, you will not see the container in the "docker ps" command. You have to use the following command:

```
$ docker ps -a
```

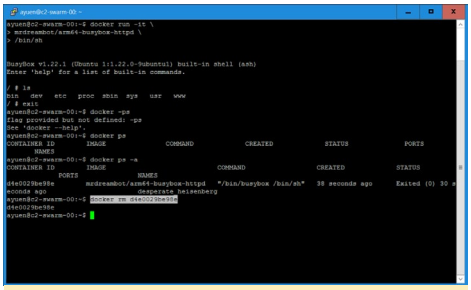
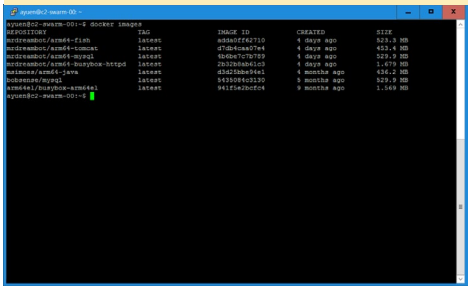


Figure 10 - docker ps -a and docker rm command outputs

I did not give a name to the interactive session. I have to identify the session and issue a "docker rm" command as shown where d4e0029be98e is the container Id identified in the "docker ps -a" command.

Figure 11 - Docker images



Managing images

You can list the images available on your Docker host using the "docker images" command as shown in Figure 11.

As described earlier, if you do not have the image on your Docker host when you start a container, Docker will try to download it from the Docker Hub. However, you can download the image beforehand using the "docker pull" command:

```
$ docker pull mrdreambot/arm64-
busybox-httpd
```

You can remove an image from your Docker host using the following command, where adda0ff62710 is the image ID listed in the "docker images" command:

```
$ docker rmi adda0ff62710
```

In Figure 11, adda0ff62710 refers to the mrdreambot/arm64-fish image.

Searching for images

Figure 12 shows two examples of searching for arm64 images. The first command searches for all images with "arm64" in the image name and limits the maximum of results to 10.

Figure 12 - Docker search

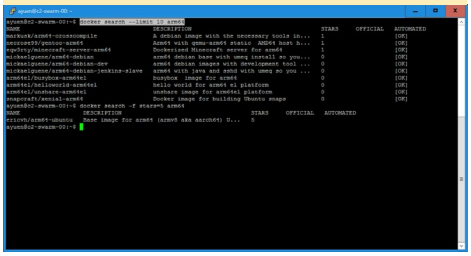
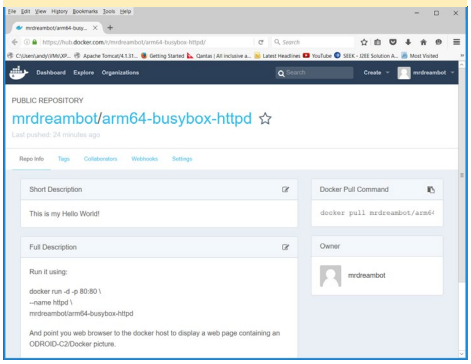


Figure 13 - image details



The second command searches for all images with "arm64" in its name and have a star rating of 5 or above. You can use the -- help option to explore other options for the search command:

```
$ docker search --help
```

You can also do a search using your browser by navigating to http://hub.docker.com. You can find out more information such as how to use the image you are interested in on the Docker hub, as shown in Figure 13.

Managing persistent storage

If you save data inside your Docker container, the data will be gone once the container is removed using the "docker rm" command. There are different ways of making your data persistent such as using Data Volumes or Data Containers. However, the easier way is to use the "-v" option as shown below:

```
$ docker run -d -p 3306:3306 \
 -- name mysql \
 -e MYSQL_USER=fishuser \
 -e MYSQL_PASSWORD=fish456 \
 -e MYSQL_DATABASE=fish \
 -v /media/sata/fish-mysql:/u01/
my3306/data \
 mrdreambot/arm64-mysql
```

-e MYSQL_USER=fishuser sets the environment variable to tell the container to set the MySQL user to fishuser

-e MYSQL_PASSWORD=fish456 sets the environment variable to tell the container to set the MySQL password to fish456

-e MYSQL_DATABASE=fish sets the environment variable to tell the container to set the database to use fish

-v /media/sata/fish-mysql:/u01/my3306/data bind mounts the /media/sata/fish-mysql volume on the container's :/u01/my3306/data directory which is the MySQL database data directory. This means that MySQL will save all data onto your host directory /media/sata/fish-mysql.

Although we have provided the storage for the database, we have not yet initialized the database content. The next section shows how this is done.

Using multiple Docker containers

In this section, I am demonstrating a more common deployment scenario in which an application has a web front end and a MySQL database backend. This means that the web front end and the database are running in their own container. For this tutorial, I am using the WEB4J sample application called the "Fish and Chips Club". From now on, I am going to refer to this application as "Fish". This application includes features to:

- edit club members
- edit local restaurants
- edit ratings of each restaurant
- add new lunches (a given restaurant on a given day)
- RSVP for each upcoming lunch
- interact using a simple discussion board
- produce simple reports
- provide a simple search page

Fish uses 3 databases running on MySQL. You can find out more about how to configure this application at http://bit.ly/2eHmxOW.

Running the Fish application

To start the Fish container, issue the following command:

```
$ docker run -d \
 -p 8080:8080 \
 --name fish \
 -e MYSQL_SERVER=192.168.1.100 \
 -e MYSQL_PORT=3306 \
 mrdreambot/arm64-fish
```

The option -e MYSQL_SERVER=

192.168.1.100 sets the environment variable to tell the container the name/IP address of the MySQL server

-e MYSQL_PORT=3306 sets the environment variable to tell the container the port number to use to access the MySQL server

Notice that up to now, we've not set up the fish databases with content yet. We are going to do that in the next section.

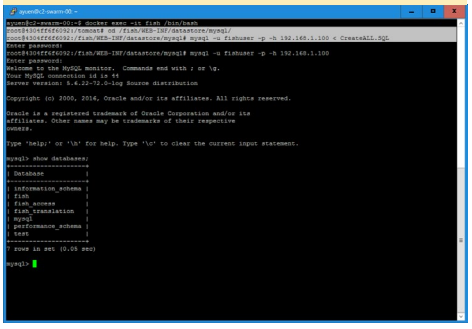
Setting up the database

To set up the Fish databases, issue the following commands:

```
$ docker exec -it fish /bin/bash
$ cd /fish/WEB-INF/datastore/
mysql/
$ mysql -u fishuser -p -h
192.168.1.100 < CreateALL.SQL
$ mysql -u fishuser -p -h
192.168.1.100
$ show databases;
$ exit
$ exit
```

The SQL script to set up the Fish databases is in the Fish container's /fish/WEB-INF/datastore/mysql/ directory. We get into the running Fish container using "docker exec" and proceed to run the MySQL client to initialize the MySQL database which runs on a separate container. After that, we use the MySQL client again to verify that the databases have been created using the "show databases;" command, as shown in Figure 14.

Figure 14 - Create database



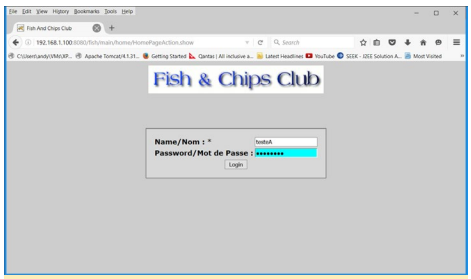


Figure 15 - Fish and Chips login

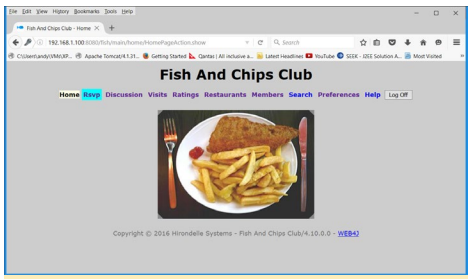


Figure 16 - Fish and Chips club

Test-driving "Fish"

Now that we have set up the databases required for Fish with proper content, we can point a web browser to the Fish application at http://192.168.1.100:8080/fish. Replace 192.168.1.100 with your ODROID-C2's IP address. You will be asked to login. Use "testeA" and "test-test" as username and password respectively. After logging in, you will see the Fish home page.

Creating a Docker image

So far, we have used images already created by me. How do we create an image in the first place? Here is a really quick look at how this is done. First, clone my simple mrdreambot/arm64-busybox-httpd on Github at http://bit.ly/2eWnLdb. Here is the content of the Dockerfile which tells Docker how the images is to be created:

```
FROM arm64el/busybox-arm64el
MAINTAINER MrDreamBot
COPY www /www
ENTRYPOINT ["bin/busybox"]
CMD ["httpd, "-f", "-p", "80",
"-h", "/www"]
```

FROM - specifies that this images is based on the arm64el/busybox-arm64el images

MAINTAINER - specifies the author of the Dockerfile

COPY - copies the www directory in the current directory to the image's /www directory

ENTRYPOINT - set default command and argument to start the container

CMD - sets additional defaults that are more likely to be changed.

If mrdreambot/arm64-busybox-httpd still shows up when you issue the "docker images" command. Use the "docker rmi" command to remove the images before you carry out the next step. Then, change to the directory where you cloned my project and issue the following command, as shown in Figure 17:

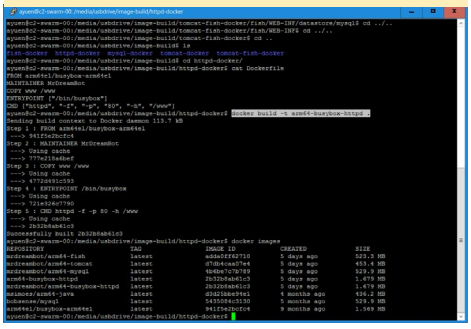
```
$ docker build -t arm64-busybox-
httpd .
$ docker images
```

Now you have created your first Docker image called arm64-busybox-httpd, which can be deployed in the same way that I showed you in the "Running your first container" section. Just replace "mrdreambot/arm64-busybox-httpd" with "arm64-busybox-httpd" in the "docker run" command and point your browser to it to see the same ODROID-C2 image.

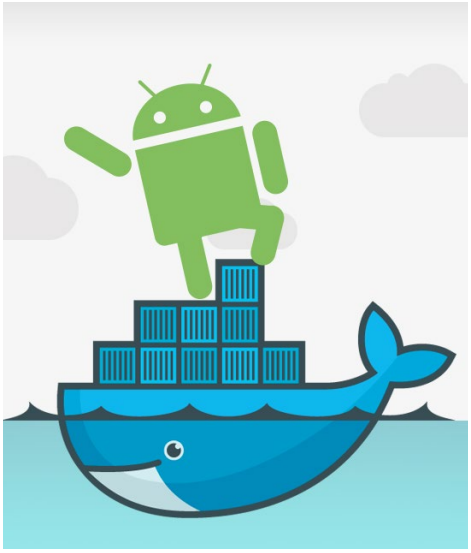
What's next?

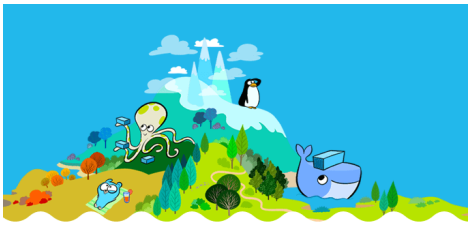
In this tutorial, I've outlined all the classic Docker commands that you will need to run and manage applications running in containers. The Docker

Figure 17 - Build Image



commands that I showed you will only work on your local Docker host, which in this case is your ODROID-C2. You will realize soon there is a limit to the number of containers that you can run on a single machine. What about enterprises that use Docker in a production environment? Surely a single machine will not be able to run all their production workload! This is where Docker Swarm Mode comes in. Swarm mode is new to Docker 1.12. It has a built-in orchestration engine, which means that, in this context, it handles clustering, scheduling of workload, and state management. Clustering is the use of a set of machines to work together and act like a single machine. Scheduling and state management means deciding where to run the containers among the machines that make up a cluster, and how many replicas of the containers should be run. Before Docker 1.12, you have to do lots of extra setup work before you can achieve Docker orchestration. In Part 2, I will show you how to use swarm mode commands for orchestration.

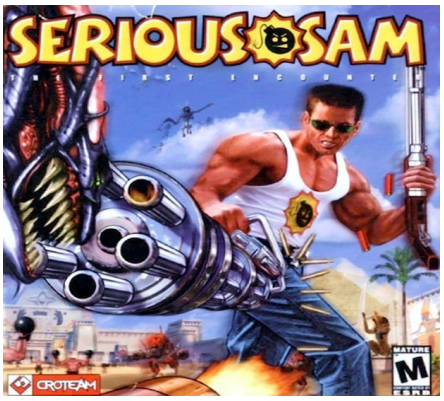




LINUX GAMING

GET SERIOUS WITH THE SERIOUS-ENGINE

by Tobias Schaaf



In this article, I want to talk about a guy that is nearly as much of a renegade as Duke Nukem himself. His name is Sam, and he's very serious! A while ago, I saw that forum user @ptitSeb from the OpenPandora forums was working on one of his many awesome game ports. This time, it was the "Serious-Engine", which is an open source engine for the game Serious Sam – The First Encounter and Serious Sam: The Second Encounter. I remember this game well, and I spent many hours with a friend fighting wave after wave of monsters. I immediately started trying to compile the games for the ODROID platform with moderate success. I got them to work, but since there was no installer with it and the structure of the game files and libraries was somewhat strange, I temporarily abandoned the project. @ptitSeb kept improving his version, as well as GLshim, which he used to run the engine, and now the game runs very well. I took the time again to compile and test the game, and was finally able to create an installer to take care of the different requirements of the game, as well as make it easier for users to add the missing game data.t

Installation

As usual, you can install this game from my repository, the installation steps of which are detailed at http://bit.ly/2eOG92v. Serious Sam is in my jessie/main package list, both games Serious Sam – The First Encounter (TFE) and Serious Sam – The Second Encounter (TSE) can be installed separately depending on which game(s) you own:

```
$ apt-get install ssam-tfe-odroid
$ apt-get install ssam-tse-odroid
```

The Serious-Engine requires OpenGL, and since ODROIDs don't have OpenGL but only OpenGL ES, we have to use GLshim to run the game, which is also provided by @ptitseb. You will also need the original game data files, specifically the "Data", "Levels" and "Demo" folders, as well as all of the ".gro" files. These files have to be placed in the game folder, which is in your home folder of the current user and named either ".tfe" or ".tse". Copy your game files in that folder and you're ready to play.

Known issues

Although the game is working very well, there are some issues that I encountered during my tests, which I hope can be fixed sometime in the future. For example, there seem to be some graphical glitches on the Exynos boards (ODROID-X, X2, U2, U3, XU3, and XU4) which show strange colors in different places. These issues are not every-

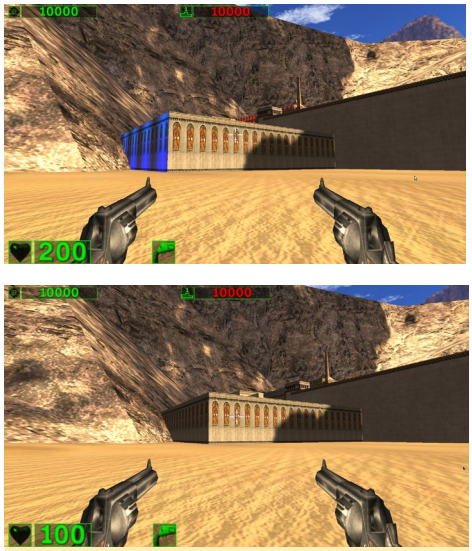


Figure 1 - ODROID XU3/XU4 (top) vs ODROID C2 (bottom), showing some of the color glitches that can be seen on Exynos devices

where, and can probably be ignored, but it's slightly annoying. I couldn't see the same glitches on the C2, so apparently it has something to do with the Exynos drivers. Although they are not pretty, these glitches don't hinder you in playing the game. They do not appear too often on the screen, and in later levels they disappear completely, so this bug can probably be ignored.

I also found that no music is working currently at this time, but I think that's an issue that can be solved, and it might already be fixed by the time this article is released. Also, it seems that the mul-

Figures 2 and 3 - Serious Sam offers a lot of nice weapons and monsters

Figure 4 and 5 - I love the great sharp shooting action in Serious Sam

tiplayer modes don't yet work. You can't join an Internet game, although there are plenty of servers available. Apparently, even LAN games are not currently working correctly, so I hope this can be fixed as well.

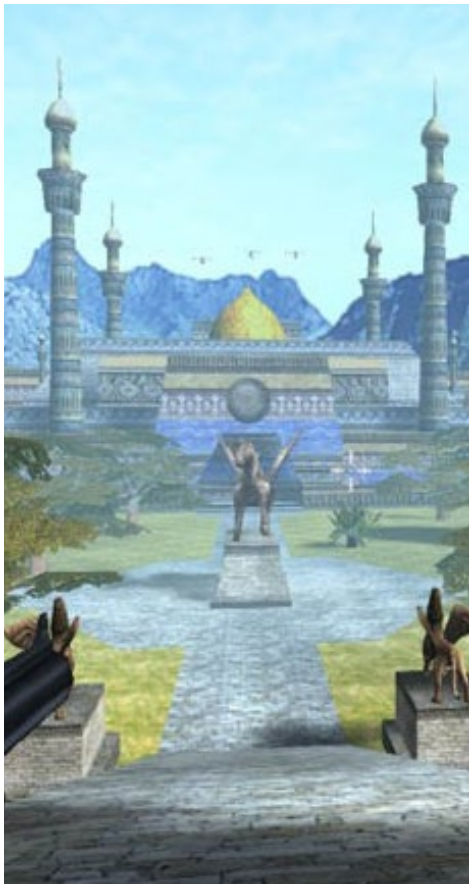
I also tried split screen, which seems to work, and you can play with up to four people with different controllers, or as a single player using mouse and keyboard. However, the C2 is not powerful enough to handle two players simultaneously, and on the XU4, I saw some graphical glitches where the picture started to flicker, which looks like a vertical sync issue. I hope that LAN gaming can be fixed, since this game is really awesome in multiplayer.

There seems to be another issue in Serious Sam – The Second Encounter as well, where you slip underneath the surface and are stuck between the level and the ground. This happens randomly and only on a few locations. If it happens, you can't do anything except reload the game and restart it from the last save point. @ptitSeb is working hard on fixing some of these issues, and I will update Serious Sam and GLshim when there are some fixes available.

Gameplay

You might ask, with all the issues, what is actually working? Well, it seems mostly everything else. You can play Serious Sam – The First Encounter and Serious Sam – The Second Encounter in single player mode with all of its goodies and baddies. It's really a one-of-a-kind shooter that defies all types of realism. You are a one man army, killing hundreds and hundreds of monsters. The game looks great and plays nicely on ODROIDs.

As you can see from the screenshots, you can still play and enjoy the game. Have fun playing the game and keep an eye on the forums for updates and bug fixes.

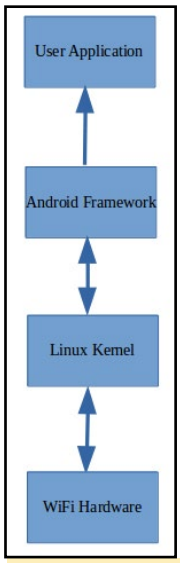


ANDROID DEVELOPMENT

ANDROID WIFI STACK

by Nanik Tolaram

Not only are Android devices typically quite powerful and come packed with features, but they can also be quite portable and easy to carry. But what good is any device without a connection to the Internet? Of course, portability means doing this without wires, which leaves us two main ways of getting our device online: via Wi-Fi or a cellular connection. All Android devices have built-in Wi-Fi functionality, and in this article we're going to take a look at how that Wi-Fi connectivity works internally inside the operating system. The source code that is used in this article is based on Android Open Source Project (AOSP) android-5.1.1_r38 release build.



A High Level Overview Android's Stack

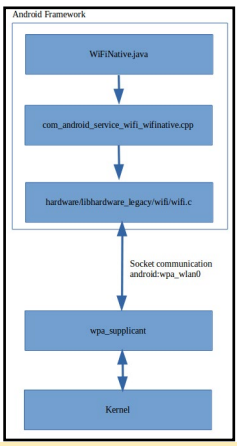
A High Level Overview

Let's start by taking a look at Figure 1. It shows a high level interaction between the different system stacks inside the Android operating system. The top layer is where our applications like YouTube and Twitter run. To make it easier to build applications, Android uses a framework to help these applications communicate with the kernel and hardware-level technology that makes our internet connection work. This 2nd layer is where we're going to take a look at the Wi-Fi stack for Android and how it works between the Linux Kernel and our applications.

Understanding wpa_supplicant

In our previous article, we looked at the HAL (Hardware Abstraction Layer) inside Android to understand how applications use this framework to communicate with the hardware inside our devices. Normally most software takes advantage of this layer, but Wi-Fi doesn't. This is because it uses an open source low-level stack to support its software-hardware communication called wpa_supplicant.

The Android framework uses wpa_supplicant as a way to communicate with the Linux kernel, much like many Linux and Unix-based operating systems. This is achieved by using the wpa_supplicant client library to communicate via a socket connection to the wpa_supplicant daemon running on the device. As shown in Figure 2, commands are initiating from an application in the Android framework that uses the socket connection to the daemon in order to relay Wi-Fi commands to the hardware itself. The daemon helps ensure that these commands, such as enabling and disabling the Wi-Fi radio, are understood by the Linux kernel.



A closer look at the Wi-Fi stack

The wpa_supplicant is initialized during Android startup process, as seen in Figure 3. This snippet was taken from http://bit.ly/2eNRphu.

```
service wpa_supplicant /system/bin/wpa_supplicant \
    -iwlan0 -Dnl80211,wext -c/data/misc/wifi/wpa_supplicant.conf \
    -e/data/misc/wifi/entropy.bin \
    -O/data/misc/wifi/sockets \
    -g@android:wpa_wlan0
    #   we will start as root and wpa_supplicant will switch to user wifi
    #   after setting up the capabilities required for WEXT
    #   user wifi
    #   group wifi inet keystore
    class main
    socket wpa_wlan0 dgram 660 wifi wifi
    disabled
    oneshot
```

The wpa_supplicant daemon startup process

Internal framework

Android user applications have access to the Wi-Fi hardware under the Linux Kernel layer through the framework by accessing the Context.WIFI_SERVICE service using the Context object:

```
$ Context.getSystemService(Context.WIFI_SERVICE)
```

The user will launch an instance of the WiFiManager providing users with the ability to access Wi-Fi services such as their current active connection, re-associating with the connection, and many more features. You can see the implementation of the "actual" Wi-Fi manager in Figure 4, inside the frameworks/opt/net/wifi directory.

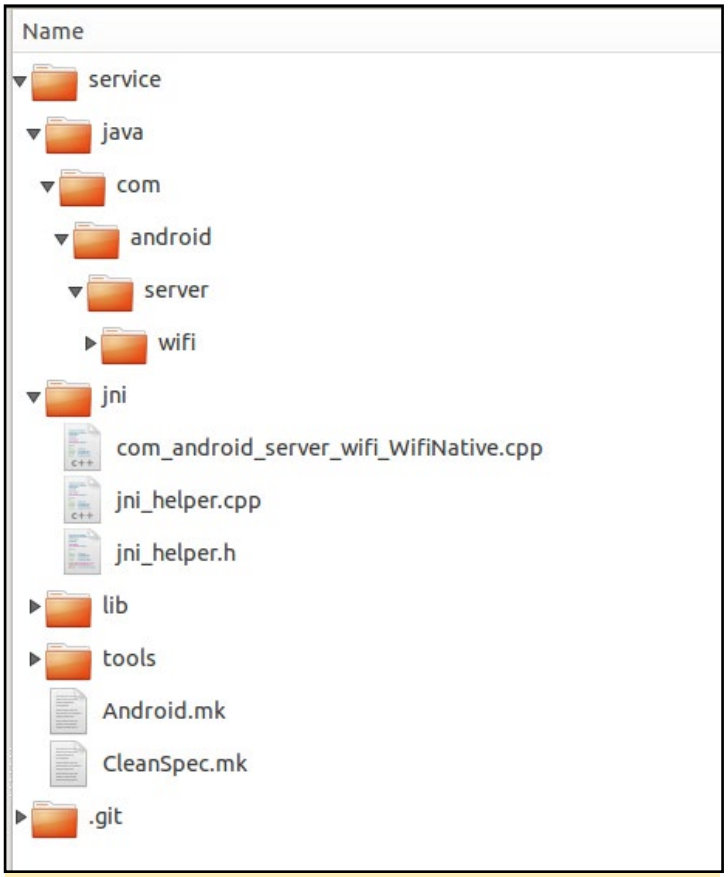


Figure 4 - The frameworks/opt/net/wifi folder

If you're interested in knowing the framework code that lives in each of these folders, you can review the information outlined in Table 1.

frameworks/opt/net/wifi/service/java/com/android/server	Contains Java code that is responsible for exposing the API (WifiManager) for user application to access wifi specific functionality, Storing wifi state management, wifi security, etc
frameworks/opt/net/wifi/service/jni	Folder containing Android source code interfacing with the wpa_supplicant library

Table I - The frameworks/opt/net/wifi directory code

The low level code that interfaces with the wpa_supplicant is packaged inside the libwifi-service.so binary file, which can be seen in the Android.mk Makefile as shown here.

```
include $(CLEAR_VARS)

LOCAL_REQUIRED_MODULES := libandroid_runtime libhardware_legacy

LOCAL_CFLAGS += -Wno-unused-parameter -Wno-int-to-pointer-cast
LOCAL_CFLAGS += -Wno-maybe-uninitialized -Wno-parentheses
LOCAL_CPPFLAGS += -Wno-conversion-null

LOCAL_C_INCLUDES += \
        $(JNI_H_INCLUDE) \
        $(call include-path-for, libhardware)/hardware \
        $(call include-path-for, libhardware_legacy)/hardware_legacy \
        libcore/include

LOCAL_SHARED_LIBRARIES += \
        libnativehelper \
        libcutils \
        libutils \
        libhardware \
        libhardware_legacy \
        libandroid_runtime \
    libnl

LOCAL_STATIC_LIBRARIES += $(LIB_WIFI_HAL)

LOCAL_SRC_FILES := \
        jni/com_android_server_wifi_WifiNative.cpp \
        jni/jni_helper.cpp

LOCAL_MODULE := libwifi-service

include $(BUILD_SHARED_LIBRARY)
```

A look at Android.mk

As you can see, the libwifi-service.so file uses com_android_server_wifi_WifiNative.cpp, which is loaded by the framework as shown in Figure 6 during the time while the class is being set up.

```
public class WifiNative {

    private static boolean DBG = false;
    private final String mTAG;
    private static final int DEFAULT_GROUP_OWNER_INTENT     = 6;

    static final int BLUETOOTH_COEXISTENCE_MODE_ENABLED     = 0;
    static final int BLUETOOTH_COEXISTENCE_MODE_DISABLED    = 1;
    static final int BLUETOOTH_COEXISTENCE_MODE_SENSE       = 2;

    static final int SCAN_WITHOUT_CONNECTION_SETUP          = 1;
    static final int SCAN_WITH_CONNECTION_SETUP             = 2;

    // Hold this lock before calling supplicant - it is required to
    // mutually exclude access from Wifi and P2p state machines
    static final Object mLock = new Object();

    public final String mInterfaceName;
    public final String mInterfacePrefix;

    private boolean mSuspendOptEnabled = false;

    /* Register native functions */

    static {
        /* Native functions are defined in libwifi-service.so */
        System.loadLibrary("wifi-service");
        registerNatives();
    }
```

A look at WifiNative.java

State Management

The Wi-Fi stack tracks every changes in the Wi-Fi connection and this is done by using state management internally. In keeping tabs with the different Wi-Fi state, this allows the framework to react accordingly and provide the ability to inform user application via broadcast intent. Below we outline the different classes we uses internally to understand Wi-Fi state management status.

Class

DefaultState
InitialState
DriverStoppingState
DriverStartingState
DriverStartedState
DriverStoppedState
ScanmodeState
SupplicantStartingState
SupplicantStartedState
SupplicantStoppingState
VerifyingLinkState
RoamingState

L2ConnectedState
WaitForP2PDisableState
SoftAPRunningState
WPSRunningState
SoftAPStartedState
ConnectedState
ObtainingIPState
DisconnectingState
DisconnectedState
ConnectModeState
TetheringState
TetheredState
TetheringState
UntheteringState

List of Wifi state management intents

Intents are the lifeblood of Android applications, and internally the framework uses each intent to communicate wifi status/states to user applications. Let's take a look at an example at how the intent is used internally to inform user applications about some state. The easiest example is when applications wants to be informed about Wi-Fi status (enabled/disabled) so it can take some action accordingly. This is done using the following <intent> declaration in AndroidManifest.xml.

```
<receiver android:name=".WifiReceiver">
   <intent-filter>
     <action android:name="android.net.wifi.WIFI_
STATE_CHANGED" />
   </intent-filter>
</receiver>
```

Inside the application there will be a class that extends the BroadcastReceiver, like this:

```
public class WifiReceiver extends BroadcastReceiver {
    …
    @Override
    public void onReceive(final Context context, final
Intent intent) {
        int wifiState = intent.getIntExtra(WifiManager.
EXTRA_WIFI_STATE, -1);
        if (WifiManager.WIFI_STATE_CHANGED_ACTION.
equals(intent.getAction())
                && WifiManager.WIFI_STATE_ENABLED ==
wifiState) {
            …
        }
    }
```

The application now all ready to receive intent from the framework when the wifi state changes. Inside the framework this intent is broadcast out in the WifiStateMachine.java class as shown here.

Internally, there is more code before calling the setWifiState(..) but this method is the point where the user will get information about the current Wi-Fi state. I hope this gives you some insight about the Wi-Fi stack and how your applications interact with it.

```
private void setWifiState(int wifiState) {
    final int previousWifiState = mWifiState.get();

    try {
        if (wifiState == WIFI_STATE_ENABLED) {
            mBatteryStats.noteWifiOn();
        } else if (wifiState == WIFI_STATE_DISABLED) {
            mBatteryStats.noteWifiOff();
        }
    } catch (RemoteException e) {
        loge("Failed to note battery stats in wifi");
    }

    mWifiState.set(wifiState);

    if (DBG) log("setWifiState: " + syncGetWifiStateByName());

    final Intent intent = new Intent(WifiManager.WIFI_STATE_CHANGED_ACTION);
    intent.addFlags(Intent.FLAG_RECEIVER_REGISTERED_ONLY_BEFORE_BOOT);
    intent.putExtra(WifiManager.EXTRA_WIFI_STATE, wifiState);
    intent.putExtra(WifiManager.EXTRA_PREVIOUS_WIFI_STATE, previousWifiState);
    mContext.sendStickyBroadcastAsUser(intent, UserHandle.ALL);
}
```

Sending WIFI_STATE_CHANGED_ACTION

MYTHTV

RUNNING THE OPEN-SOURCE HOME ENTERTAINMENT APPLICATION ON YOUR ODROID-C2

by @WebMaka

I use several ODROID-C2s on my TVs as front-ends for watching live video streams, using MythTV as the main software driver. On the back-end is a standard computer running Ubuntu 16.04.1 LTS and using a HDHomeRun PRIME with a paired CableCard and tuning adapter as the means of converting my digital cable service into streams that I can view remotely. Since this can take a lot of time to set up the first time, here's a how-to on creating the same setup for yourself. This guide is written for the latest LTS release of Ubuntu as of its writing, 16.04.1 LTS, but I know it works on 14.04 LTS as well and should also work on any Debian derivative.

About DRM

DRM is the bane of live TV users everywhere. Expect this to be your single biggest problem, and if you're trying to wean others off live TV, such as your spouse or elderly parents, expect the biggest point of contention to be over not being able to watch DRMed content or channels.

Please note that this means MythTV cannot and will not work with any live-TV programming that is flagged as "protected/copy-once" or "protected/no-copy." There is, by design, no way around this with any open-source software. The whole point of this approach is to force the use of "approved" hardware and software that you have to spend money month after month to use, under the premise that this protects content from redistribution.

Some cable companies mark almost everything as copy-once, but others only mark premium paid-sub channels, like HBO or Cinemax. Some channels that, in my opinion, should not be marked as copy-once, are often done because of demands to that effect on the part of the channel, not the cable company. If your cable company flags everything or nearly everything as heavily restricted, don't bother with HDHomeRuns or any other consumer-owned equipment. You'll have to stick with the cable box or just stop using cable television service entirely.

The HDHomeRun Prime can view protected content with the right software, but if you do want to watch and stream DRM-protected content, the only sure way of doing this is to use Windows Media Center running on Windows. Keep in mind that only Windows 7 includes Windows Media Center for free. SiliconDust reportedly had an Android app that did allow watching copy-once content but the current app doesn't do it consistently. I did hear that you can watch DRMed content from a HDHR Prime over an Nvidia Shield, but I haven't tried this myself.

Setup

Here are the general steps necessary to use MythTV with your ODROID:

1. Install and update Ubuntu 16.04 LTS
2. Install HDHomeRun driver software on your Linux box
3. Install and configure the MythTV backend on your Linux box
4. Set up the clients/frontends on your ODROID
5. Watch live TV along with anything you can download/stream over the Internet

Install and update Ubuntu

If you're using the latest 16.04 LTS version of Ubuntu already, start off by doing repetitions of the following two commands until everything's up-to-date:

```
$ sudo apt-get update
$ sudo apt-get upgrade
```

If you're using an older version, keep in mind that upgrading the OS is usually more time-consuming and more risky than just reinstalling fresh from the latest image. These instructions should work to varying degrees with other Debian-based distros, although your experience may differ.

Install HDHomeRun driver

To install the HDHomeRun libraries and configuration tool, use the following command:

```
$ sudo apt-get install libhdhome-
run hdhomerun-config
```

Next, install a suitable driver. The easiest approach for the driver part is to grab the DVB driver with Debian packaging that was published on GitHub by @h0tw1r3:

```
$ git clone https://github.com/
h0tw1r3/dvbhdhomerun
$ cd dvbhdhomerun
$ dpkg-buildpackage -b
```

If this fails with a dependency error, you'll need to run "sudo apt-get install" for whatever packages are missing, such as dkms. Assuming that the buildpackage command completed successfully, let's install the built packages:

```
$ cd ..
$ dpkg -i *hdhomerun*.deb
```

Once this is done, you should be able to launch the HDHomeRun Config GUI and have it list your Prime's three tuners. If it doesn't detect the HDHR Prime, correct this before proceeding, since you may have skipped a step or had a failure that you overlooked.

Install and configure MythTV's backend

Type the following command to install the MythTV package:

```
$ sudo apt-get install mythtv
```

Expect a number of dependencies to be installed on a fresh Ubuntu install, so, let it do what it needs to do. Then, click the Ubuntu button at the top of the bar and then click MythTV Backend Setup. You may need to search for it if it's not right there on the first row of icons.

If it asks for information for a database connection, note what the settings say for database name, username, and password and then close the backend configuration application. You'll need to add the database to MySQL, create the user account that MythTV will use, and give it full permissions on the database MythTV will use. Do not just give it the MySQL "root" username and password, as this is a very insecure method. Instead, use a MySQL management tool to create a new user for MythTV. Then, restart the backend setup program. If all went well, it shouldn't ask for database connection details. If not, correct this before proceeding.

It's important to give the machine that will be the primary backend for MythTV a static IP address. You will need to access it at a fixed IP, and DHCP will most likely change the IP address whenever the router reboots. Consult your router's documentation on how to assign IPs statically. Do this before proceeding if the machine currently has a dynamic/DHCP-assigned IP.

Setting the IP address

Once MythTV's backend setup tool is happy with the MySQL configuration and its ability to connect to the database, you should get a series of options with "General" as the first option. "Host Address Backend Setup" is where you'll start. Change "IPv4 Address" to whatever the static IP is. Do not leave it on 127.0.0.1, or it will only accept client connections from localhost, effectively ignoring all other clients on the network.

Change "IPv6 Address" to blank unless you're using IPv6. If you don't do this, MythTV may bind only to an IPv6 address and may ignore IPv4 connections. It's not supposed to, but it did for me and people have complained online about this happening to them. Then, change "Security Pin (required)" to 0000 (for "allow any clients," or to an actual pin if you need to restrict access) or MythTV won't accept connections.

For the "Master Backend" part, change "IP address" to match the static IP as well, or again it'll only watch localhost for connections from slave backends if you use them. Keep pressing "next" until you end up back at the main menu. For the initial setup, you shouldn't need to change anything else in the General options, and can go back later if you do.

Setup the capture cards

Navigate to the "Capture Cards" section. We'll do this three times, one for each tuner on the HDHR Prime. First, click "New Capture Card", or use the arrow keys to select and hit Enter. For the first box, "DVB Device," open the list and choose the HDHomeRun option. If you do not have it, you missed something in step one, so backtrack accordingly. Again, if HDHomeRun Config cannot "see" your HDHR Prime, MythTV won't either. In "Available Devices," open the list and pick the first tuner. It'll be an ID code for the HDHR followed by "-0", e.g., 1234ABCD-0. "Tuner" should be "0" for obvious reasons. Click "Recorder Options."

Increase "Signal timeout (ms)" to "10000" (read: add zero at the end) and increase "Tuning timeout (ms)" to "30000" (see previous). This will reduce timeouts from slow clients or during overloads on busy non-gigabit networks. Reduce "Max recordings" to "1" as the HDHR Prime can only support one action at a time on any one tuner.

Click "Next" and "Finish" until you see the list of capture cards, and repeat the setup for the other two tuners on the HDHR Prime. Then, press the Escape key to get back to the main menu.

Configuring the video sources

In the main menu, navigate to "Video Sources." This is where you'll set up your channel listings and guide. Give

the guide setup a suitable name in "Video Source name."

If you have an active subscription to SchedulesDirect.org for listing/guide data, change "Listings Grabber" to the SchedulesDirect option and fill in the blanks for the username and password as required. Then, click "Retrieve Lineups" to fetch what you've selected in your SchedulesDirect.org account for your location. Choose the appropriate entry for "Data Direct lineup". MythTV will then fetch channel/guide data as required. Just make sure to leave "Perform EIT scan" unchecked, since EIT data will overwrite all over the SD.org guide data and your guides will be much less usable.

If you don't have an active sub to SchedulesDirect.org, consider spending the $25 a year for one, as their guides are very useful. At the very least, get a 7-day trial to see if their data will suit your needs. Otherwise, if you're not in an area they have listings for, leave "Listings grabber" on "Transmitted guide only (EIT)" to try to pull down the cable company's guide information or change it to "xmltv Selections" and step through the options for that if you have another grabber setup that you'd like to use.

Again, click "Next" and "Finish" until you return to the list of sources. Repeat the steps if you want to set up more than one, and press the Escape key to the main menu when you're done.

Input connections

Navigate to the "Input Connections" section, where you tell MythTV to use the guide source's channel, frequency, and program data to tell the tuners what to tune to. Please note that we will also do this part three times, once for each tuner, but with a couple subtle but important changes. Click the first HDHR tuner on the list, or use the arrow keys and the Enter key. I left "Display name" blank but you can change it to anything so long as it's unique for your stream.

For "Video Source," pick the grabber you had set up previously, such as your SchedulesDirect.org account. For "Use quick tuning," I didn't see where it made any difference for my HDHR Prime, so I left it on "Never."

Click the "Fetch channels from listings source" button and immediately hit the down-arrow key twice. It'll take a few seconds or so to fetch the channel/freq/program data, and you'll know it's finished when the selected control jumps suddenly to the "Next" button. When it does, click the "Next" button.

This is an important step: "Schedule Order" and "Live TV Order" will be different for each tuner. Tuner 0 will be "3" for "Schedule Order" and "1" for "Live TV Order", tuner 1 will be "2" for both, and tuner 2 will be "1" for "Schedule Order" and "3" for "Live TV order" 3 and 1, 2 and 2, and 1 and 3. If you don't do it this way, you'll have clients fetch tuners out of order and likely won't have HDHR hand out access to all three tuners because MythTV will think it's at the end of the tuner pool before it actually hands out all three. Also, if you don't give them a reverse sequence between the two settings, MythTV will hand out tuner 2 to the first client and the HDHR Prime will tell MythTV it's out of available tuners. This took me a lot of time to figure out, since it's not mentioned anywhere except for a single forum post that I stumbled across.

I left the input group settings set to the default values. Press "Next" and "Finish" until you return to the list of sources, and repeat the steps if you want to set up more than one. Use the Escape key to return to the main menu when you're done.

From the main menu, hit the Escape key again. MythTV should then display a dialog about refreshing the guide if you changed any channel info. Click OK or tap Enter to dismiss this and let the configuration program close. Confirm that you want to run the MythTV backend, and give it the proper username and password. Then, run "mythfilldatabase" so that it will fetch the complete channel lineup and programming guide. The backend will come up pretty quickly, which can be tested by navigating a browser to http://localhost:6544. The guide update will take approximately 20 minutes to run the first time.

Set up the clients

Most people that user MythTV backends are probably going to be using Kodi for the clients. I'm running Kodi 16.1 (Jarvis) on a custom build of LibreELEC for the ODROID-C2 since OpenELEC's support for the C2 is not as robust. This is very likely going to change when OpenELEC hits its first full 7.x release, since C2 support is currently in development.

Find the MythTV PVR plugin in Kodi by navigating to System -> Settings -> Add-Ons -> My Add-Ons -> PVR Addons, then finding the plugin and selecting "Config". Give it the IP address of the backend machine, enable the add-on by following the same navigation steps and selecting "Enable", then restart Kodi. The plugin should then fetch the channel list/lineup/guide info from the backend and the "TV" section should be visible in the main menu.

Watch live TV

From your device and environment of choice, select "TV," select "Guide," and select a channel/show. If everything is working properly, you should be able to enjoy your video content on the ODROID-C2. For comments, questions, or suggestions, please visit the original thread at http://bit.ly/2dvs3oQ.

ANDROID NOUGAT

IMPRESS YOUR FRIENDS WITH THE LATEST ANDROID VERSION

by @voodik

Android Nougat, which was officially released earlier this year, introduces notable changes to the operating system and its development platform, including the ability to display multiple apps on-screen at once in a split-screen view, support for inline replies to notifications, as well as an OpenJDK-based Java environment and support for the Vulkan graphics rendering API, and "seamless" system updates on supported devices.

Features

- Android 7.0 Nougat Cyanogenmod 14.0
- Kernel 3.10.9
- OpenGL ES 1.1/2.0/3.0 (GPU acceleration)
- OpenCL 1.1 EP (GPU acceleration)
- Multi-user feature is enabled (Up to 8 users)
- On board Ethernet and external USB 3.0 Gigabit Ethernet support
- RTL8188CUS , RTL8191SU and Ralink Wireless USB dongle support
- USB GPS dongle support
- USB tethering
- Portable Wi-Fi hotspot
- Android native USB DAC support
- USB UVC Webcam support
- HDMI-CEC support
- Selinux Enforce
- Bluetooth and USB-3G don't work yet

Installation

To install Android Nougat onto a blank media, you need prepare your eMMC or SD card with an appropriate self-installation image, which is available at http://bit.ly/2eWb1zi. Write the image to your eMMC or SD card using Win32DiskImager (http://bit.ly/1Vk9u4o).

If you are updating from the CM 13.0 image, copy and paste the following URL into the ODROID-Updater URL section:

http://oph.mdrjr.net/voodik/5422/ODROID-XU3/Android/CM-14.0/Alpha-0.1_10.10.16/update.zip

The update process might take up to 20 minutes, so be patient.

Tips

To get Wifi working set the correct module name in the build.prop file. For example, to use a Realtek 8192cu (default), set the following parameter:

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

To use Realtek 8188eu:

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

To use Ralink RT33XX/RT35XX/RT53XX/RT55XX:

```
wlan.modname=rt2800usb
```

If you are using a USB GPS dongle, set the correct tty port and speed in build.prop:

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

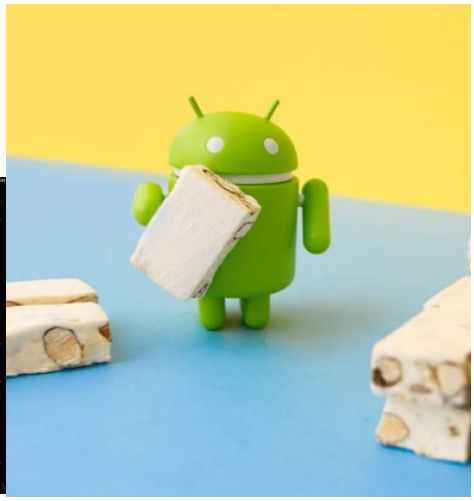
To enable the ability to shutdown the system without confirmation by long pressing the power button, add the following line:

```
setprop persist.pwbtn.shutdown
true
```

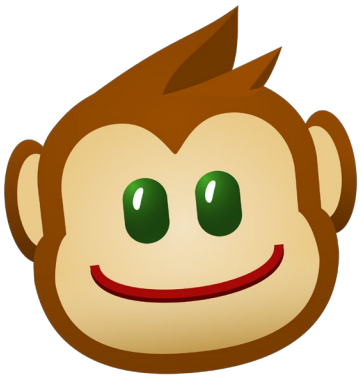
If you want to enable the Cloud Print feature, download and install Google Cloud Print Service app from http://bit.ly/2e2YyLw.

For comments, questions or suggestions, please visit the original post at http://bit.ly/2ec67zI.

Figure 1 - Android Nougat screenshot



ACCELERATED VIDEO PLAYBACK FOR BROWSING ON THE ODROID-C2



WATCH YOUR WEB MEDIA CONTENT IN FULL HD

by Adrian Popa

If you're using your ODROID-C2 as a desktop computer, chances are you're also surfing the web on it. By now, you may have noticed that in-browser video playback doesn't have the performance that you would expect. Although you can view 720p Youtube videos in-browser, it's choppy, so 360p is the only acceptable resolution where playback is watchable. Other sites simply report that they can't play any videos.

In order to improve video playback quality, two things are needed: Accelerated video decoding, which is handled by the aml-libs package on the C1/C2, and accelerated rendering, which should be handled by the X11 drivers. Unfortunately, Chrome does not support accelerated video decoding for aml-libs though, so playback is done with the CPU instead, causing these slow and choppy experiences on HD video streams. The goal of this article is to offload the video playback to a process which can do it using accelerated hardware and improve the browser's video playback experience

The best approach

The best way to get this done is to fix the egg before we have a problem with the chicken in the first place. By this, we mean adding aml-libs support to Chrome's source code. If you look into Chromium's documentation (http://bit.ly/2eTfcww), you can see that a custom-built ffmpeg package is used for video decoding. This means that as soon as some brave developer ports aml-libs support to ffmpeg, we might have Chromium builds that play accelerated videos on C1/C2. Work is being done with patches to ffmpeg by forum user @LongChair that should allow native playback in the end (http://bit.ly/2eWjRwY). Hopefully, by the time you're reading this article, this approach will already be working.

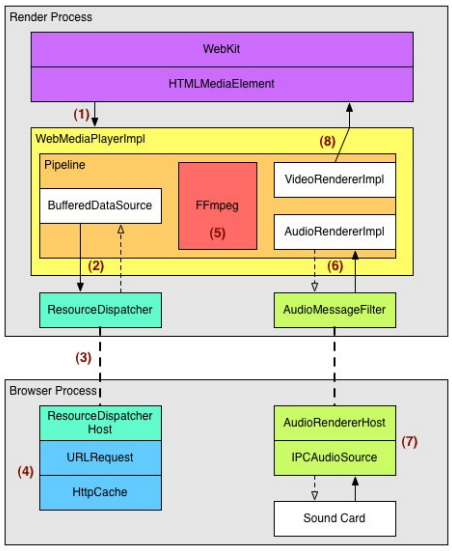


Figure 1 - A diagram of the rendering process

The convoluted approach

In case the "best approach" method is not yet ready, we'll have to try something else. The technique in this guide is inspired by work done on Firefox by a friend of mine. A word of caution is that it might not be directly applicable to Chromium due to changes in Chromium's plugin API (and its deprecation of NAPI). So, the story goes that my friend has a very old PC that he uses to browse the web. Unfortunately, the PC could not keep up with playing back video content in the browser, but it could do the job in an external player, so my friend, whose name is Silviu, decided to do something about it instead of throwing money at the problem. He used Firefox and the Greasemonkey plugin to write his own mini-plugin for Firefox and managed to offload video playback to the mpv video player software. He did this all while rendering everything in the browser window itself to give the impression of a cohesive experience. This is what I wanted to try to replicate on ODROIDs at first.

What he did was use Greasemonkey to load a custom 1600-line JavaScript function that overrides all HTML5 video elements and replaces them with proxy objects that forward the calls to his plugin. For this he needed to reimplement the whole HTML5 video API which is described at https://mzl.la/2dRRTjP and https://mzl.la/2eTgUxS. His code gets the calls the browser makes to create new video objects, set the video source, and can start, seek, and stop playback.

Upon intercepting the messages, the JavaScript code creates a request with a fake content-type which is handled by a plugin library specially designed to be the middleman in this scenario. The JavaScript code doesn't have the right to read or execute files, but the plugins do, so the plugin is used to start up an mpv instance and communicate with it through a piped connection. The JavaScript code sends play, pause, and seek requests to the plugin, which it then converts them into mpv commands and lets the mpv process do all the heavy lifting. Also, the reverse is true: when you control mpv through its built-in controls, it reports things such as play/pause/seek events back to the plugin library, which translates them back to JavaScript events which are passed through the Greasemonkey plugin back to the original player.

The result of all this is that the original player shows the correct status, current position and can be used to control the player in an intuitive way. This is important because a lot of sites have built-in protections against scraping and may query periodically through JavaScript to see the player's state and check for anything strange taking place.

In terms of presentation, using a plugin allows you to create a new operating system window without titlebar decorations, and position it on top of the HTML5 video object. The fun fact is that the window is embedded inside the webpage so it gets hidden out of view when you scroll. MPV can use the --wid parameter to draw its output to an arbitrary window, which is a cool option that works in Windows and Linux alike. This makes the player look like it's native and unobtrusive.

Figure 2 - MPV plays to a Mousepad editor window

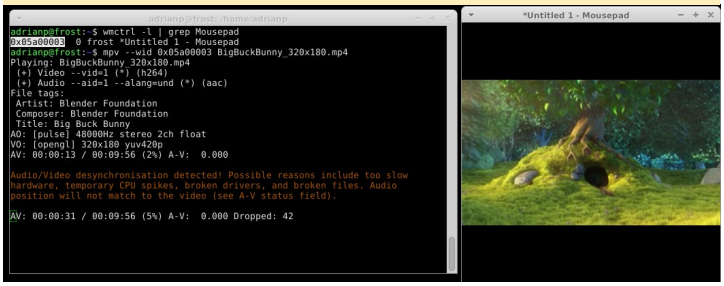
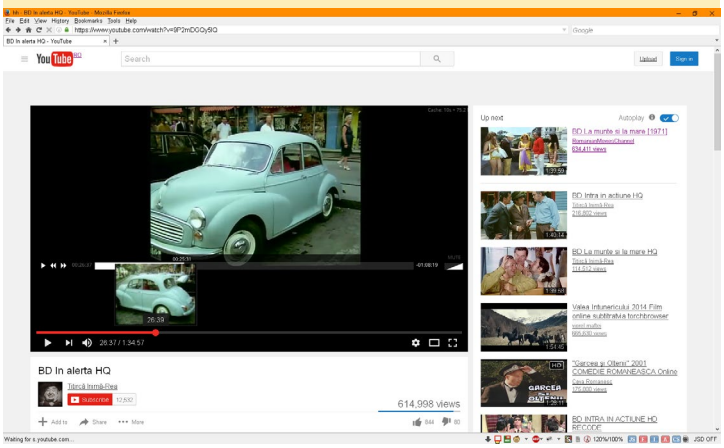


Figure 3 - Firefox proof-of-concept with skinned mpv controls



Also, the plugin allows you to create multiple instances of the player, so that several tabs can play video simultaneously.

Unfortunately, the solution I've seen was tested only on Firefox and Windows, but with a bit of work, it could be ported to Chromium as well. I have to convince my friend to release his sources so that others could join in.

The resulting solution

A relatively simple way to get accelerated playback from the browser is to pass the URL of the video to a player which has aml-libs support. Fortunately, after a long and very entertaining discussion on the forum at http://bit.ly/2eLzOpA, we now have a standalone player for the C2 as well as patches to ffmpeg and mpv that could lead to a more diverse player selection in the future. However, for our needs, we'll use c2play, a minimalistic player built especially for the C2 by forum user @crashoverride (http://bit.ly/2eXOp0s).

Our plan consists of the following: 1) create a chrome plugin to send the current URL (or the video element's URL) to a backend script, and 2) the backend script calls youtube-dl to get the video URL (if needed) and calls the player to play it.

Fortunately, Chrome has a native messaging API (http://bit.ly/1cOVcrU) that lets you communicate with external processes (which it's kind enough to spawn for you), so this is what we'll do.

To install c2play, you will need to consult the latest instructions from the ongoing support thread at http://bit.ly/2eXOp0s, since development is progressing rapidly. But at the time of this writing, you can install it with the following commands:

```
$ git clone -b beta1 https://github.com/OtherCrash-
Override/c2play.git
$ sudo apt-get install libasound2-dev \
  libavformat-dev libass-dev libx11-dev premake4
$ cd c2play
$ premake4 gmake
$ make c2play-x11
$ sudo cp c2play-x11 \
  /usr/local/bin/c2play-x11
```

You'll also need youtube-dl, which you can get from Ubuntu's repositories, or from http://bit.ly/1Vej3Vi. Youtube-dl is a program that takes a page URL and extracts the video element URL embedded inside. It can do this for a wide range of websites that either use HTML5 or Flash for playback (http://bit.ly/2d9yknp). I recommend that you install it manually, because changes to sites happen often and you'll need to be able to update it easily, which can be done with the following set of commands:

```
$ sudo curl -L \
  https://yt-dl.org/downloads/latest/youtube-dl \
  -o /usr/local/bin/youtube-dl
$ sudo chmod a+rx \
  /usr/local/bin/youtube-dl
```

Finally, to install the Chrome plugin that ties all of these together, follow these steps:

```
$ git clone https://github.com/mad-ady/odroid.
c2.video.helper.git
$ cd odroid.c2.video.helper
$ sudo apt-get install libjson-perl \
  libproc-background-perl libconfig-simple-perl
```

To install the plugin, you will need to navigate inside Chrome to chrome://extensions/. Next, drag and drop from a file browser the extension located in odroid.c2.video.helper/release/odroid.c2.video.helper-1.1.crx (or version 1.2, as described below) into the Chrome window, and it will prompt you to install it.

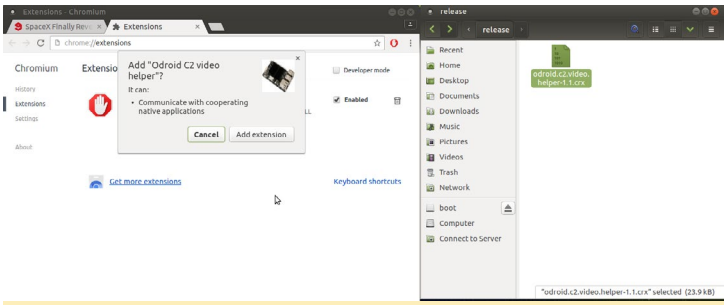


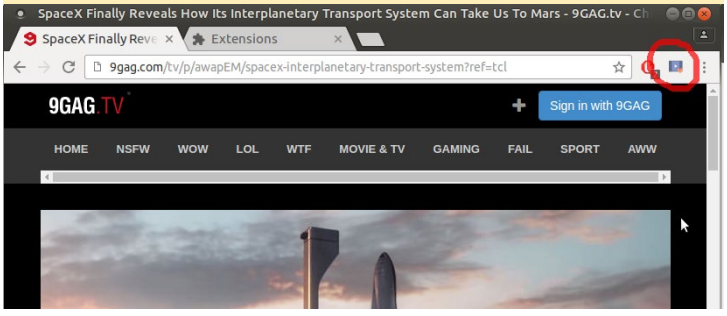
Figure 4 - Installing our plugin

Once it is installed, continue to install the backend script:

```
$ cd host
$ bash install_host.sh
```

The install_host.sh script creates the necessary directories in your Chromium installation and copies over the backend script (typically to $HOME/.config/chromium/NativeMessagingHosts) and its configuration file to your home directory. At this point, you should see a new button that looks like a video player next to your address bar. If you can make a better icon, please contribute to the support thread referenced at the end of the article.

Figure 5 - Push the button!



After activating the plugin, when you navigate to a website that has a video embedded, you can press this button, and the tab's URL will be passed to the backend script. The backend script will run it through youtube-dl, which can take about 5-6 seconds, and will obtain the URL of the video. It will then call the preferred player with this URL, and playback should start. A video of the installation and playback process is available at https://youtu.be/Z3ng8Qm8STI.

At the time of writing, I only tested it with c2play-x11 with playback in fullscreen. In order to control it, consult the help page at http://bit.ly/2eWl5IV. Adding windowed playback support is on @crashoverride's todo list (http://bit.ly/2fd8C7a) and will probably become a reality in the near future. Also, @LongChair's mpv port could be used for playback in the future. For now I've tried with the stock mpv, and playback is fine for SD content, but 720p seems to lag. You can view a demo at https://youtu.be/fJLv0cwPyfg.

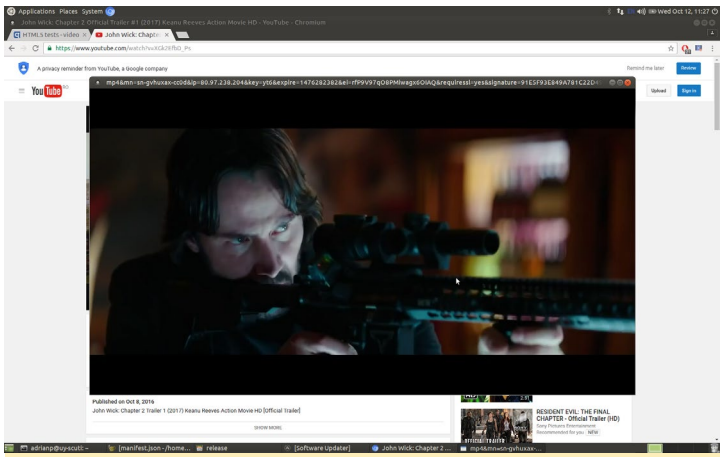


Figure 6 - Non-accelerated playback in mpv

The backend script has a configuration file stored at ~/.odroid.c2.video.helper.conf using an ini syntax. With this file, you can disable debugging, which is on by default. You can also change the player and set custom parameters, such as quality, for various sites supported by youtube-dl. For instance, the default config file sets the youtube quality to 720p (-f 22) and disables playlist support. You can add new sections, which must have a name that matches the URL's domain. Also, the section name must not contain the character dot (".") because it's not well supported by the configuration parser module. To view debugging information you can run:

```
$ sudo journalctl -f
```

Playing 1080p or 4k from YouTube is a bit more problematic because, for resolutions higher than 720p, YouTube splits video and audio into two distinct streams, and their player

```
adrianp@uy-scuti:~$ cat .odroid.c2.video.helper.conf
[general]
debug=1
playerdebug=1
player=/usr/local/bin/c2play-x11

[youtu]
;set quality to 720p
extraArgs="-f 22 --no-playlist"
;set quality to 3840x2160 with audio mp4a.40.2@128k (44100Hz)
;extraArgs="-f266,140 --no-playlist"

[vimeo]


adrianp@uy-scuti:~$
```

Figure 7 - A configuration sample

merges them on the client browser. This doesn't happen on other sites such as Vimeo. Fortunately, @crashoverride also has a branch called dualstream that tries to support this, as described at http://bit.ly/2fd7RLt. Preliminary tests show that he is on the right track. I was able to play 4K video with his player, but playback unfortunately stalled after several seconds. This will likely improve in the near future.

What should you expect to work and what should you expect not to work? Sites supported by youtube-dl, such as YouTube, Vimeo, Facebook, IMDB, Engadget, DailyMotion, TED, Cracked, Apple Trailers, 9gag TV and a variety of adult sites should all work. In order to play content, you need to navigate to a page with a single video on it and click on the button next to the address bar. However, sites which require login to view the content will not work, nor will sites which require cookies to access the data, due to how the plugin connects and processes the stream.

Relying on youtube-dl for site support is fine, but my idea was to retrieve the URL to the video directly from the browser's DOM. In case you're not aware of how things work in the wild Internet, web pages can create video objects that have various properties including a source pointing to the video being played. Youtube-dl uses page scraping and lots of internal magic to extract this URL and present it to other processes. However, since we're running inside the browser ,we should be able to get the source of the video, thus saving about 6 seconds of processing time. I modified the Chrome extension to look for video objects, and the plan was to create a button which would initiate playback externally. But this is guaranteed to fail, since each page embeds the video under various layers of objects, most of which are either hidden or overlap with other ones, so the button would be hidden or positioned incorrectly in the page.

My next attempt was to intercept the "playing" event and pause the video, steal the URL and pass it to the backend for playback. This worked surprisingly well for my test site (http://bit.ly/1nAXG0z), but failed for larger sites such as YouTube and Vimeo. Digging through the page revealed why: these sites use a technique called "media-source" (http://bit.ly/2eWmfnG) which uses JavaScript to make requests and stream the video, so we can't extract a URL to pass for playback.

For sites that use a video element with a real URL as as the source, the plugin will play the content with the external player on the first playback attempt. If you try to play the video again without reloading the page, it will play normally inside the page. This is a protection in case the external player can't handle the video format and you want to fall back to the browser. In this case, you should just see a black window flash briefly on the screen while playback is attempted, otherwise it will play normally.

I have released two versions of the plugin. Version 1.1 supports only playback through youtube-dl through the use of the toolbar button. Version 1.2 adds direct playback experimental support, but may not always work, or may be too annoying for videos which play automatically when you enter the page, so which plugin you use is up to your needs.

I'd like to hear your suggestions and feedback on the support thread at http://bit.ly/2eBVtCZ. All this wouldn't have been possible without the hard work of @crashoverride and @LongChair, so send them a big thank you!

Figure 8 - Normal video object vs media-source

```
    crossOrigin: null
    currentSrc: "http://www.quirksmode.org/html5/videos/big_buck_bunny.mp4"
    currentTime: 12.766221
```

```
crossOrigin: null
currentSrc: "blob:https://www.youtube.com/1155dccf-f07e-4967-86de-6e5ded2ad7a8"
currentTime: 3.889002
```

MEET AN ODROIDIAN

JOACHIM ALTHOF

edited by Rob Roy

Joachim sitting on the roof of a wind turbine in Estonia during a commissioning trip in April 2015

Please tell us a little about yourself.

My name is Joachim Althof, and I am 32 years old. I live near Hanover in Germany along with my wife Katrin and our two daughters. I did my studies in Mechatronics at the University of Applied Sciences in Lemgo, which is a tiny town in northwestern Germany. After finishing my studies, I worked for a company near the border to Denmark, where I wrote embedded microcontroller software for relatively large grid inverters (>100 kW) and other kinds of photovoltaic- and wind-power converters. Five years later, I went back to my former hometown and started to work for a new company. I am the first, and so far only, software developer for small PMSM inverters (<10 kW), and am responsible for many other things regarding development coordination.

How did you get started with computers?

I gained my first computer experiences with my father's Commodore 64 back in the early 1990s. Thinking back, this was really old-school, with an almost round-shaped greenscale monitor that flickered a lot, , a reset-button and homemade loudspeaker, along with a 9-pin parallel-port printer with gray "endless paper". At first, I had absolutely no clue about what was going on when I typed <load "$",8,1>. And honestly, I didn't really care, as long as the games started. A short while later, I started to be more interested in understanding what was going on inside of that gray box. I even got GEOS running, which was a type of graphical OS for the C64, and managed to install the printer driver. When I was around 12, I got my first i486 computer running DOS. Due to the fast hardware development at that time, that device was slowly upgraded piece by piece and became my first "serious" computer for years. At home, my wife says that I am "responsible for everything with a cable".

What attracted you to the ODROID platform?

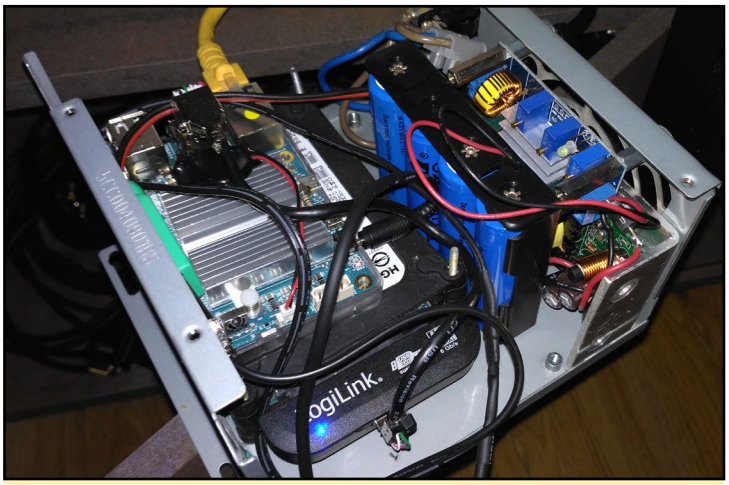
When I started with SBCs, the Raspberry Pi was already widespread. I wanted to join the hype, but desired something different and more powerful than the RPi. I decided to purchase a Cubieboard2, which was brand-new those days. I learned a lot and met some really nice people in the forum. However, I was soon slightly disappointed by the company, the community and the product, so I searched for a new platform to play around with, and that's how I discovered the ODROID platform. I was impressed by the hardware specs and the reasonable price, so I bought my first ODROID, which was the C1 model. I ported all of my projects onto it and it was fun to see what it can do, and also to discover all its quirks.

I was very satisfied with the performance, because it did everything that I wanted it to do. When the C2 came out, I said to my wife, "Look at this! We need this one! More is better!". Admittedly, she didn't really care, but I bought my first C2 and a little bit later, my second C2 arrived, accompanied by my wife shaking her head at me.

I have found that the ODROID community is a very active one. Of course, there are some more and some less active members, but in general, I was happily surprised by the community. What I highly appreciate is the fact that even the administrators and Hardkernel members contribute to the forum. For me, it looks like there is an active collaboration between the Hardkernel members and the internet community, which is priceless. The best board is worthless without proper support.

How do you use your ODROIDs?

My first approach was to get all of my projects running on one single board. My original C1 was a network file server, a media server, a media player, SVN server, a gaming machine, Ambilight-maker, a development board, and an orbital defense cannon. In Germany, we would call it an "egg-laying wool-milk-sow"! However, every time I tinkered a little bit too hard, several things broke at the same time, which prompted a low Wife Acceptance Factor (WAF). I decided to decentralize my system, which resulted in the need for more ODROIDs, which prompted my wife to say "Wait, ... what was THIS thing for?!" Now I have my C1 as a dedicated server device, which will be ported to a V2 over time, one C2 as a media player running LibreELEC, and one C2 to play around with. Later I will use one of the ODROIDs as a retro-gaming platform again.



Joachim's home server powered by an ODROID-C1, all tightly fitted into an old ATX power supply case

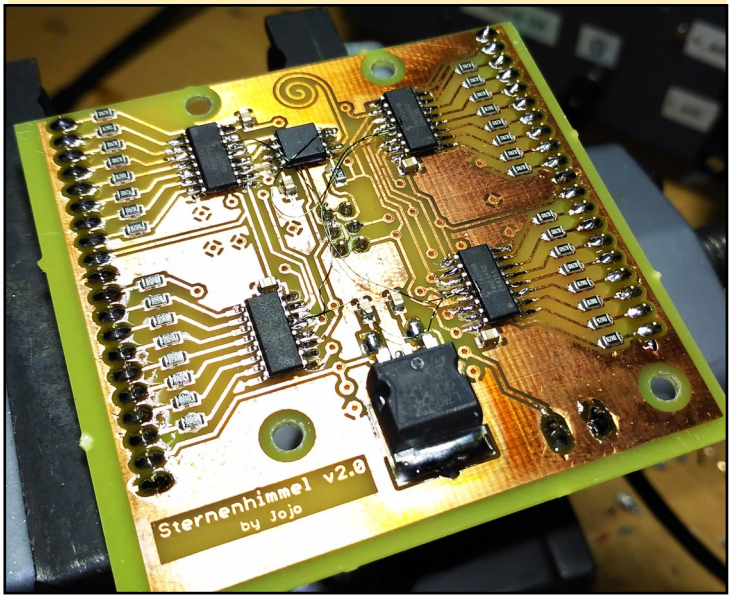
Which ODROID is your favorite and why?

At the moment, I only have experience with the C1 and C2, but my favorite is the C2. It is really powerful for the price, has a small form factor, is versatile, and well supported. In my use cases, both boards perform very well. I think there will be even more projects and possibilities, once the 64-bit ARM architecture becomes more mature. I also like the I/O header of the C1 and the C2, because it makes hardware tinkering easy. Whether it is necessary to have the I/O header be Raspberry Pi-compatible is another story.

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

That's a good question, because many features are already there. I'd like onboard-Wifi, or maybe USB 3.0, which is already available on the XU4. The most important thing to me is a good support of the features, since I had to struggle with the buggy USB drivers on the C1. Also, something similar to the RPi Zero would be great. I know that Hardkernel offers the C0, but this is not comparable with the RPi Zero in many ways. I think some people could use a MIPI-CSI port as well, especially when it is compatible with the RPi. This could open up a huge new field for ODROID applications.

A homemade PCB for a 32-channel PWM, powerd by a ATTiny85 via SPI and a bunch of shift-registers for a ceiling lamp in his daughter's room to make it look like a night sky full of sparkling stars



What hobbies and interests do you have apart from computers?

I like doing sports like jogging and Freeletics. I am a fan of metal music, and play drums. My singing is also not bad, and I like to go to music concerts and festivals. During the dark and cold days of winter, I work on an embedded project with AVR microcontrollers and a lot of soldering. Most importantly, I enjoy doing nothing as well as spending free time with friends and my family.

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

It depends on the specific kind of programming. I think that learning embedded C is a difficult way to start, because it is less "visual" than writing software for a PC. You can only see or measure your programming results on hardware I/Os. However, it's worth the effort when you think about the possibilities! Start with something simple like an LED blink loop. Don't be afraid of pointers*, although they're not necessary for beginning programmers. Always keep in mind how powerful software can be. The software is the soul of a device, and to bring life to a piece of dead hardware can be awesome! Also, knowing that something can explode by setting a wrong bit can be very exciting and scary. On the other hand, programming shell scripts or C# can be good to start with, because they is more visual then embedded programming. It really depends on what the application is. Most important: is to just do it! Show some self-initiative and people will help you as you help yourself.

Joachim and his girls having some fun at the fair, although they enjoyed it more than he did