Deluge Bittorrent Client • Chatbot • Portable Arcade Station • Clipgrab

# ODROID Magazine

Year Four
Issue #37
Jan 2017

## ODROIDS *Around the World*

We celebrate the reach of our gadgets spanning the globe in the portable computing revolution

• IoT Environmental Wine Cellar Preserver and Notifier

• Setup a rear view camera for your bike using the oCAM

# 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 ODROID-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

Suche – Pollin Electronic
www.pollin.de/shop/suchergebnis.html?S_TEXT=odroid&log=internal
DuckDuckGo
Elektronik, Technik und Sonderposten zu Superpreisen!
mein Konto
Kontakt & Service
Versand & Bezahlen
Katalog
Newsletter
Gutscheine
Merkliste | Neu registrieren | Anmelden
Direktbestellung
Einkaufswagen
Suchen nach: Suchbegriff oder Bestellnummer
Los
Kategorie wählen
Filtereinstellungen
[-] alle Filter entfernen
Kategorie
[+] Bausätze / Module (17)
[+] Stromversorgung (1)
Preis
Ihre Vorteile
Innovative Produkte
Optimales Preis-/Leistungsverhältnis
Kompetente Beratung
Schnelle Lieferung
Noch Fragen?
Wir sind gerne auch persönlich für Sie da!
+49 (0) 8403 / 920-920
Sicher Einkaufen
Suchergebnis
Ihre Suche nach "odroid" ergab 18 Treffer
sortieren nach: Relevanz (gewählt)
1 - 18 von 18 Ergebnissen
Seite: 1 von

ODROID-U3 BACKUP BATTERY
RTC Lithium-Backupbatterie für den ODROID XU, XU+E, XU Lite und U3. Wird direkt auf den RTC-Port des ODROID gesteckt.
2,95 €* Best.Nr. 810 319

ODROID-U3 Einplatinen-Computer, Cortex-A9 QuadCore, 2 GB
Kleiner als eine Scheckkarte aber riesig in der Leistung! Der ODROID-U3 verfügt über eine leistungsstarke SAMSUNG QuadCore-CPU, 2 GB Arbeitsspeicher und
69,95 €* Best.Nr. 810 264

ODROID-U3 ODROID-UPS, USV, LiIon, 3 Ah
Unterbrechungsfreie Stromversorgung mit zwei LiIon-Akkus. Das Board besitzt die gleichen Außenmaße wie der ODROID-U3 Einplatinencomputer und kann direkt per
31,95 €* Best.Nr. 810 328

ODROID-U3 eMMC Modul, 64 GB, mit Linux
Superschnelles NAND-Speichermodul für den ODROID-U3 Einplatinencomputer. Das Linux-Betriebssystem XUBUNTU ist fertig aufgespielt und kann direkt

ODROID-U3 USB-UART MODULE, Schnittstellenkarte
USB/UART System-Interface (IC: CP für die ODROID Einplatinencomput X2, U2, U3, XU, XU+E, XU Lite). I Entwicklung und Debugging.
15,95 €* Best.Nr. 810 326

ODROID-U3 CASE Kunstst
Passgenaues Gehäuse mit Lüftungsschlitzen für den Maße (LxBxH): 87x52x29
4,95 €* Best.Nr. 810 317

ODROID-U3 eMMC Android
Superschnelles N den ODROID-U3 Betriebssystem aufgespielt un
39,95 €* Best.Nr. 81

ODROID-U3 Linux
Supersch den OD Linux- berei

ODROID Magazine is now in its 4th year! We are very excited to continue presenting community-contributed articles and projects that highlight the versatility and portability of Hardkernel's fantastic line of single board computers. Some of the projects that we have featured in the past year include a water-cooled ODROID-XU4, an Ambilight 4K system, a 42" touchscreen table, and a portable laptop. We look forward to seeing what innovative and unique projects ODROIDians will create in 2017 and beyond.

IoT projects have become a very popular use of ODROIDs, and our resident IoT expert Miltiadis presents a project that combines two of his pastimes, wine and computers, to monitor his wine cellar in order to make sure that his valuable collection is aged properly. The system also notifies him via SMS when the wines are ready to enjoy. Max details how to set up a chatbot, Brian demonstrates a rear view bike camera to stay safe while riding, @synportack24 gives an overview of the Deluge BitTorrent client, and Tobias shows us how to download online videos for offline viewing. Our featured Android games this month include Pixel Dodgers and Chrome Death for hours of fun!

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

ameriDroid.com
High-Performance Embedded Computers

Hundreds of products available online for the professional developer and hobbyist alike

ODROID-XU4

ODROID-C1+

ODROID-C0

OWEN ROBOT KIT

ODROID-C2

VU7 TABLET KIT

Hardkernel's EXCLUSIVE North American Distributor

## Rob Roy, Chief Editor

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

## Bruno Doiche, Senior Art Editor

It has passed four years, and Bruno still holds his b eer glass high and make you wonder: Isn't that beer now hot? Didn't he tire of holding it for so long? Or is it just a picture?

## 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, a number of ODROID-U3's, and Xu4'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.

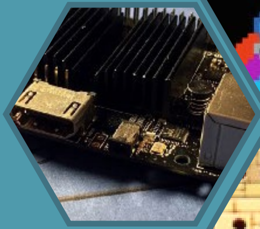
## Venkat Bommakanti, Assistant Editor

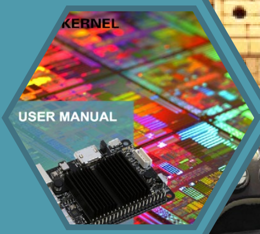
I'm a computer enthusiast from the San Francisco Bay Area in California. I try to incorporate many of my interests into single board computer projects, such as hardware tinkering, metal and woodworking, reusing salvaged materials, software development, and creating audiophile music recordings. I enjoy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





# IOT ENVIRONMENTAL WINE CELLAR PRESERVER AND NOTIFIER

by Miltiadis Melissas (@miltos)

Any good wine preserver knows how important it is to store wine in conditions as close to a traditional wine cellar as possible in order to ensure the taste and quality of the fine beverage. Those conditions are based on the temperature, the humidity and the lighting of the space used as a wine cellar. In particular, the temperature must be held as close as possible to 12 degrees Celsius, though fluctuations between 10 and 14 degrees are acceptable, in order to guarantee a good aging process. With respect to humidity, it must be at least 50% to ensure a warm and moist environment for the wine. Lastly, it must be a dark room as often as possible, as light can cause oxidation and unnaturally accelerate the aging process. All of these elements must be in a delicate balance to ensure your wine ages beautifully.

This project is the capstone of my previous 3 articles regarding the Internet of Things (IoT) using an ODROID-C2. This IoT device, the environmental wine cellar preserver and notifier, so to speak, monitors the fermentation conditions in the wine cellar using a temperature, humidity, and light sensor. The most important feature in this project is the ability to receive an SMS alert if ideal fermentation conditions are not being met, however the fermentation data can also be transmitted daily to the cloud for remote monitoring as well. In this project, we'll be using a cloud-based communication (PaaS) company for the proper delivery of those SMS messages (i.e Twilio). Please refer to our previous article published on Hardkernel's October's issue (http://bit.ly/2fFXJHQ) for details on how to use such a service. An LED is also added to the device which is controlled programmatically, indicating to a user physically in the room whether ideal conditions are in the wine cellar at that moment.

Figure 1 - A Wine Cellar is an opportunity ripe for fermentation

## Building the circuit

We will use a breadboard to hold our electronics together just as we did with our previous IoT project, the street and the home lights controller with SMS notifier (http://bit.ly/2fFXJHQ), in order to avoid any soldering and the hassle of designing a PCB for our prototype. We will connect various circuit components with the ODROID-C2 GPIO pins using Dupont Jumper Wires, as shown in Figure 2. Here's a list of all the hardware and software we'll be using for this project:

## Hardware:

- ODROID-C2 running Ubuntu 16.04
- 5V/2A Power Supply from HardKernel (http://bit.ly/1X0bgdt)
- Breadboard and Dupont male to female jumpers
- A DHT11 Temperature and Humidity Sensor
- A Photo resistor
- (2) Resistors (4.7K and 220Ω)
- A 1 μF Capacitor
- An RGB LED

## Software:

- Ubuntu 16.04 v2.0 from Hardkernel (http://bit.ly/2cBibbk)
- Python 2.7 or 3.3 (preinstalled

on Ubuntu)

- WiringPi Library for controlling the ODROID-C2 GPIO Pins. You can learn how to install this at http://bit.ly/2ba6h8o

## Building the IoT device

For our wired 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 male side connects into the holes of the Breadboard. Please refer to Hardkernel's pin layout schematic as you create the connections, which is also available at (http://bit.ly/2aXAlmt). Physical Pin 1 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 Pin 6 as the common Ground, we connect that to the first vertical line of our Breadboard, near the edge. The DHT11 sensor has three pins, so we connect Pin 1 to 3.3V on the board, Pin 2 (brown wire) in the middle to ODROID-C2, Pin 7, and the last one (Pin 3) to the common Ground. This is visualized in Figure 2. Next, the photoresistor/photocell is connected to physical Pin 18 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. 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 Pin 18. Finally, the operational LED is connected to physical Pin 16 for its anode (+) while the cathode (-) is connected of course to the common Ground through a 220Ω resistor. That's it! All of our physical wiring is now connected.

Before connecting anything to your ODROID-C2, disconnect the power. It is important to note that you can destroy your ODROID-C2 with a short circuit from a wrong connection. Just be careful and double check everything before powering it back on.

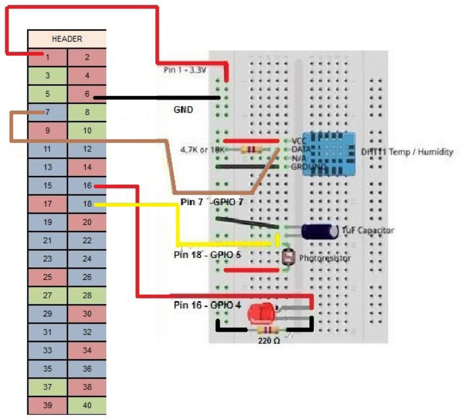


**Figure 2 - A closer look at the breadboard breakdown**

## Writing the Code

We divided the code into sections (chunks of code) for easier reading. We start by importing the necessary modules. Then we define the pins on the ODROID-C2 that we are going to use. On the next step, we set up the wiringPi module according to Hardkernel's map guide (http://bit.ly/2aXAlmt). We proceed by setting up the operation LED. Next, we interface the photoresistor and the DHT11 sensor with the ODROID-C2. Of course, before doing that, we define the variables needed for controlling them in Python. The next chunk of code is important as we are pulling data from the DHT11 sensor. Thankfully enough, there is a piece of code available in GitHub at http://bit.ly/2gAaUfK for doing the job. Since we are going to use the Twilio service, we set that up next. Last, but not least, we check for the right conditions in the wine cellar. The temperature should be around 12 degrees Celsius, the humidity at least 50%, and the absence/presence of light. The last piece of code is merely for validation purposes. Please refer to the block diagram in Figure 3.

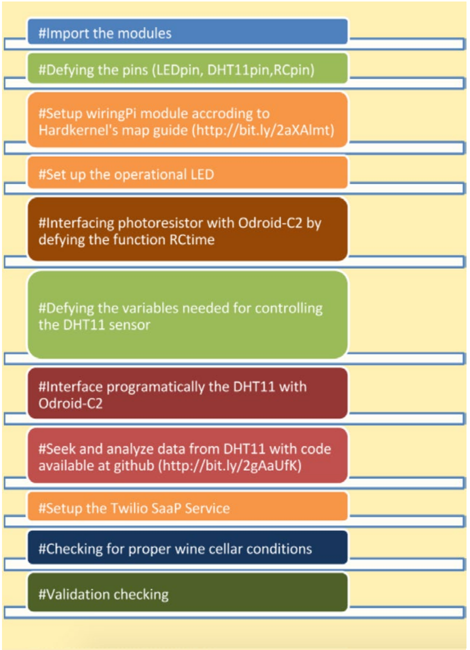

#Import the modules
#Defying the pins (LEDpin, DHT11pin,RCpin)
#Setup wiringPi module accroding to Hardkernel's map guide (http://bit.ly/2aXAlmt)
#Set up the operational LED
#Interfacing photoresistor with Odroid-C2 by defying the function RCtime
#Defying the variables needed for controlling the DHT11 sensor
#Interface programatically the DHT11 with Odroid-C2
#Seek and analyze data from DHT11 with code available at github (http://bit.ly/2gAaUfK)
#Setup the Twilio SaaP Service
#Checking for proper wine cellar conditions
#Validation checking

**Figure 3 - A breakdown of the various code chunks and related hardware in use**

## Using Twilio

Please refer to the article in the ODROID Magazine November issue (http://bit.ly/2fFXJHQ) for information on how to set up an account for Twilio and how to use this cloud based communication (PaaS) service. What we actually need are the API keys (account_sid, auth_token) for using this service and a python script for triggering SMS messages to the user when certain conditions are being met. The following piece of code does exactly this:

```
def sent_SMS():
    from twilio.rest import
TwilioRestClient

    account_sid = "xxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxx" # Your
Account SID from www.twilio.com/
console
    auth_token  = "xxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxx"  # Your Auth
Token from www.twilio.com/console

    client =
TwilioRestClient(account_sid,
auth_token)
    message = client.messages.
create(body="Alert!!! The condi-
tions on the wine cellar is out
of range!",
    to="+xxxxxxxxxx",    # Re-
place with your phone number
    from_="+xxxxxxxxxx")   # Re-
place with your Twilio number

    print(message.sid)
```

## Connecting Twilio with the photoresistor

```
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 milli-
second per loop cycle
    while (odroid.
digitalRead(RCpin) == 0):
          reading += 1
    return reading
   ...
def printData():
   ...
   #Check the lighting conditions
in your wine cellar
   if  (RCtime(5)<2500):
       print ("Alert!!! Light in
the wine cellar!")
       #send_SMS()
```

Connecting the photoresistor element with Twilio was described fully in our previous article, "ODROID-C2 as an Iot device: Street and Home Lights controller with an SMS notifier" in ODROID's November magazine (http://bit.ly/2fFXJHQ). The critical part is to define the right threshold of light through which the SMS message is triggered by calling the Twilio function. You should test and find this very specific level of light according to your wine cellar's own light conditions through trial and error. Since we don't have a wine cellar, we tested the photoresistor under the normal conditions in our room, and we find that such an SMS message is triggered by a value under 2500:

```
< #Check the lighting conditions
in your wine cellar
   if  (RCtime(5)<2500):
        print ("Alert!!! Light in
the wine cellar!")
        send_SMS()>
```

## Connecting Twilio with the DHT11 sensor

Here's the snippet we'll use with the temperature and humidity sensor:

```
def pullData():
#{{{ Pull data from GPIO.odroid
    global data
    global effectiveData
    global pin

    data = []
    effectiveData = []

    odroid.pinMode(DHT11pin,1)
    odroid.
digitalWrite(DHT11pin,1)
    time.sleep(0.025)
    odroid.
digitalWrite(DHT11pin,0)
    time.sleep(0.14)

    odroid.pinMode(DHT11pin,0)
    odroid.
pullUpDnControl(DHT11pin,2)

    for i in range(0,2900):
        data.append(odroid.
digitalRead(DHT11pin))

    """
    for i in range(0,len(data)):
        print "%d" % data[i],
    print
    """

#}}}

 ...
```

```
def printData():
   global Humidity
   global Temperature

   print "H: "+Humidity
   print "T: "+Temperature
   #Checking here if the condi-
tions in the Wine Cellar is met
   if (int(Humidity)<50) or
((int(Temperature)<10) or
(int(Temperature)>14)):
      print ("Alert!!! Tem-
perature and/or Humidity out of
range!")
        #sent_SMS()
  ...

#}}}
```

We connect the DHT11 to the ODROID-C2 using the wiringPi library, and the way we do it is not much different than the method used with the photoresistor. Since both of those elements are sensors, we directly read data from their input pin and store it in an variable or in an array. In the case of the DHT11, we are storing data in an array and this line of code does exactly this:

```
< data.append(odroid.
digitalRead(DHT11pin))>
```

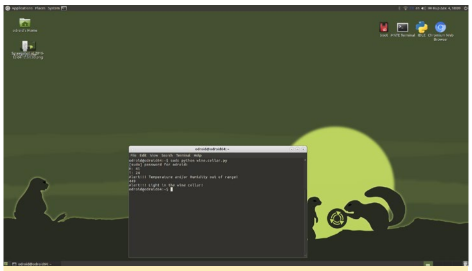
Here's a very important note: the speed at which we read the data from the DHT11 sensor will significantly influence the success or failure of the sensor. After some trial and error, we found the right value according to ODROID-C2 frequency clock. The next line of code does the sampling. The value of 2900 gave us 100% accuracy in every DHT11 reading, but there is a range of values (2900-3300) that you can experiment with if you wish:

```
<for i in range(0,2900):>
```

Later, somewhere in the end of the code, we extract the values of the humidity with the temperature, and only then do we test with the right conditions in the wine cellar before we trigger the SMS message, if necessary:

```
< #Checking here if the condi-
tions in the Wine Cellar is met
   if (int(Humidity)<50) or
((int(Temperature)<10) or
(int(Temperature)>14)):
      print ("Alert!!! Tem-
perature and/or Humidity out of
range!")
       sent_SMS()
       for i in range (0,10):
           #disable LED
         odroid.
digitalWrite(LEDpin, 0)
         # wait 1 second
         time.sleep(1)
         #enable LED
         odroid.
digitalWrite(LEDpin, 1)
           time.sleep(1)
   #cleanup
   odroid.pinMode(LEDpin, 1)
   #Print data
   print RCtime(5)  # Read RC
timing using pin #18
   #Check the lighting conditions
in your wine cellar
   if  (RCtime(5)<2500):
        print ("Alert!!! Light in
the wine cellar!")
        send_SMS()
  for i in range (0,10):
           #disable LED
         odroid.
digitalWrite(LEDpin, 0)
         # wait 1 second
         time.sleep(1)
         #enable LED
         odroid.
digitalWrite(LEDpin, 1)
           time.sleep(1)
   #cleanup
   odroid.pinMode(LEDpin, 1)
>
```

Notice also how the LED blinks each time an SMS alert message is sent, notifying the user the proper functionality of



```
odroid@odroid64:~$ sudo python wine.cellar.py
[sudo] password for odroid:
H: 41
T: 24
Alert!!! Temperature and/or Humidity out of range!
449
Alert!!! Light in the wine cellar!
odroid@odroid64:~$
```

**Figure 4 - A look at the python code running on the device to alert us**

the IoT device. However, it requires the user's personal presence.

## Bringing it all together

Let's assemble everything in Python. We'll need to first import the modules:

```
#!/usr/bin/python

import wiringpi2 as odroid
import time
import sys
```

Next, we'll define the pins as LEDpin=4 (physical Pin 16) and DHT11pin=7 (physical Pin 7):

```
LEDpin=4
DHT11pin=7
```

Then, we'll set up the WiringPi module according to Hardkernel's map guide:

```
odroid.wiringPiSetup()
```

And now we'll set up the operational LED:

```
#Set the operational LED
odroid.pinMode(LEDpin,1)
odroid.digitalWrite(LEDpin,1)
```

We interface the photoresistor by defying the function RCtime(RCpin):

```
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 milli-
second per loop cycle
    while (odroid.
digitalRead(RCpin) == 0):
          reading += 1
    return reading
```

We define the variables needed for controlling the DHT11 sensor:

```
def bin2dec(string_num):
    return str(int(string_num,
2))

data = []
effectiveData = []
bits_min=999;
bits_max=0;
HumidityBit = ""
TemperatureBit = ""
crc = ""
crc_OK = False;
Humidity = 0
Temperature = 0
```

We interface the DHT11 with Odroid-C2 via GPIO (wiringPi):

```
def pullData():
#{{{ Pull data from GPIO.odroid
    global data
    global effectiveData
    global pin

    data = []
    effectiveData = []

    odroid.pinMode(DHT11pin,1)
    odroid.
digitalWrite(DHT11pin,1)
    time.sleep(0.025)
    odroid.
digitalWrite(DHT11pin,0)
    time.sleep(0.14)

    odroid.pinMode(DHT11pin,0)
    odroid.
pullUpDnControl(DHT11pin,2)
```

```
    for i in range(0,2900):
        data.append(odroid.
digitalRead(DHT11pin))

    """
    for i in range(0,len(data)):
        print "%d" % data[i],
    print
    """

#}}}
```

This section seeks and analyzes data from the DHT11. This code is pulled from Github (http://bit.ly/2gAaUfK) as previously described:

```
def analyzeData():
#{{{ Analyze data

#{{{ Add HI (2x8)x3 bits to array

    seek=0;
    bits_min=9999;
    bits_max=0;

    global HumidityBit
    global TemperatureBit
    global crc
    global Humidity
    global Temperature

    HumidityBit = ""
    TemperatureBit = ""
    crc = ""

    """
    Snip off the first bit - it
simply says "Hello, I got your
request, will send you tempera-
ture and humidity information
along with checksum shortly"
    """
    while(seek < len(data) and
data[seek] == 0):
        seek+=1;

    while(seek < len(data) and
data[seek] == 1):
         seek+=1;

    """
    Extract all HIGH bits'
blocks.  Add each block as sepa-
rate item in data[]
    """
    for i in range(0, 40):

        buffer = "";

        while(seek < len(data)
and data[seek] == 0):
            seek+=1;


        while(seek < len(data)
and data[seek] == 1):
            seek+=1;
            buffer += "1";

        """
        Find the longest and the
shortest block of HIGHs.  Aver-
age of those two will distinct
whether block represents '0'
(shorter than avg) or '1' (longer
than avg)
        """

        if (len(buffer) < bits_
min):
            bits_min =
len(buffer)

        if (len(buffer) > bits_
max):
            bits_max =
len(buffer)

        effectiveData.
append(buffer);
        #print "%s " % buffer

#}}}
```

Now let's set up the Twilio PaaS service:

```
def sent_SMS():
    from twilio.rest import
TwilioRestClient

    account_sid = "xxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxx" # Your
Account SID from www.twilio.com/
console
    auth_token  = "xxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxx"  # Your Auth
Token from www.twilio.com/console

    client =
TwilioRestClient(account_sid,
auth_token)
    message = client.messages.
create(body="Alert!!! The condi-
tions on the wine cellar is out
of range!",
    to="+xxxxxxxxxx",    # Re-
place with your phone number
    from_="+xxxxxxxxxx")    # Re-
place with your Twilio number

    print(message.sid)
```

And now the most critical part of this project: we check the ideal conditions in the wine cellar, and if those conditions are not met, we trigger the SMS:

```
#{{{ Print data
def printData():
   global Humidity
   global Temperature

   print "H: "+Humidity
   print "T: "+Temperature
   #Checking here if the condi-
tions in the Wine Cellar is met
   if (int(Humidity)<50) or
((int(Temperature)<10) or
(int(Temperature)>14)):
      print ("Alert!!! Tem-
perature and/or Humidity out of
range!")
       sent_SMS()
       for i in range (0,10):
           #disable LED
         odroid.
digitalWrite(LEDpin, 0)
         # wait 1 second
         time.sleep(1)
         #enable LED
         odroid.
digitalWrite(LEDpin, 1)
          time.sleep(1)
   #cleanup
   odroid.pinMode(LEDpin, 1)
   #Print data
   print RCtime(5)   # Read RC
timing using pin #18
   #Check the lighting conditions
in your wine cellar
   if  (RCtime(5)<2500):
        print ("Alert!!! Light in
the wine cellar!")
        send_SMS()
        for i in range (0,10):
           #disable LED
         odroid.
digitalWrite(LEDpin, 0)
         # wait 1 second
         time.sleep(1)
         #enable LED
         odroid.
digitalWrite(LEDpin, 1)
           time.sleep(1)
   #cleanup
   odroid.pinMode(LEDpin, 1)
#}}}
```

Finally, let's do some validation checking:

```
#{{{ Main loop

while (not crc_OK):
    pullData();
    analyzeData();
    if (isDataValid()):
        crc_OK=True;
        print "\r",
        printData();
    else:
        sys.stderr.write(".")
        time.sleep(2);
#}}}
```

## Testing and running the code

We run the whole code as a final test:

```
$ sudo python wine.cellar.py
```

It should work like a charm. The program displays on the screen the temperature and humidity, and reports the presence or the absence of light in the wine cellar. It also notifies the user by sending an SMS to their phone if the proper conditions are not met. Please note that during the sending of messages, the operational LED is blinking, as an indication of the "alert" process. Next, let's automate it and make it run every 3 hours. For this task, we will use the cron utility.

What is cron? It defines jobs that are used to schedule tasks and scripts, such as deftags, backups and alarms. We used the "cron" utility previously when we designed the Gmail notifier IoT project back in the October issue of ODROID Magazine (http://bit.ly/2dwqXJ7). If you need further information about cron, please refer to http://bit.ly/2bTmNaN. In order to activate cron, we must execute the command crontab which gives us a list of scheduled tasks:

```
$ sudo crontab -e
```

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

```
* */3 * * * sudo python /home/
odroid/wine.celllar.py
```

The five "stars" ("* * * * *") specify how often you want the task to be run. The first star controls the minutes. The second star controls the hours, which is why I put a "/3" symbol after it, since I want this scheduled task to run every 12 hours. The third specifies the day of month, the fourth indicates the month, and the fifth represents the day of the week. Those four were intentionally left blank without any "/numbers" besides the stars. You can experiment with other options as well. At the end of the scheduled task, there is the command itself we want to be run automatically:

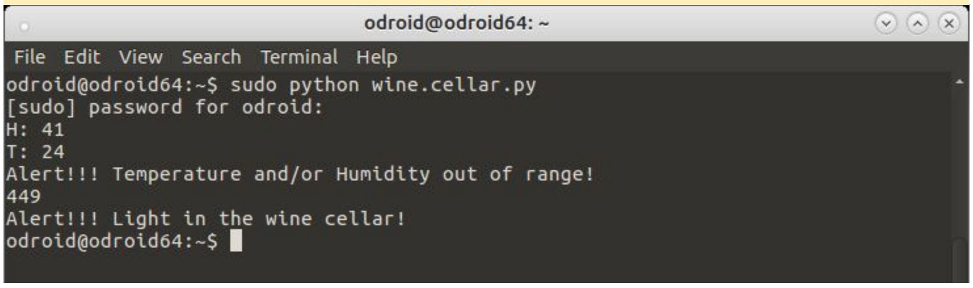
```
$ sudo python /home/odroid/wine.
cellar.py
```

This command runs our script and points to the path where it is located, which is, in this case, /home/odroid/. Then, save and close the editor. Now, wait and watch as the application does its magic. You will have a new incoming SMS message every time the temperature, the humidity, or the lighting is out of range. That's it, we did it!

## Final thoughts

The code of the wine cellar preserver and notifier can be improved or altered in many ways. One method is to add to it a feature that reports the values of humidity, the temperature and the lighting in every reading to the cloud. We have described how this can be done using Twython API in our article in ODROID Magazine's September issue (http://bit.ly/2cIyp36). Please refer to this article on how to utilize such a service. By using Twython, we can send tweets to our Twitter account with the reading values of the humidity, temperature and the lighting.

Figure 5 - A screenshot of the program running in a terminal



```
odroid@odroid64:~$ sudo python wine.cellar.py
[sudo] password for odroid:
H: 41
T: 24
Alert!!! Temperature and/or Humidity out of range!
449
Alert!!! Light in the wine cellar!
odroid@odroid64:~$ 
```

# DELUGE

## YOUR NEW FAVORITE BITTORRENT CLIENT

by @synportack24

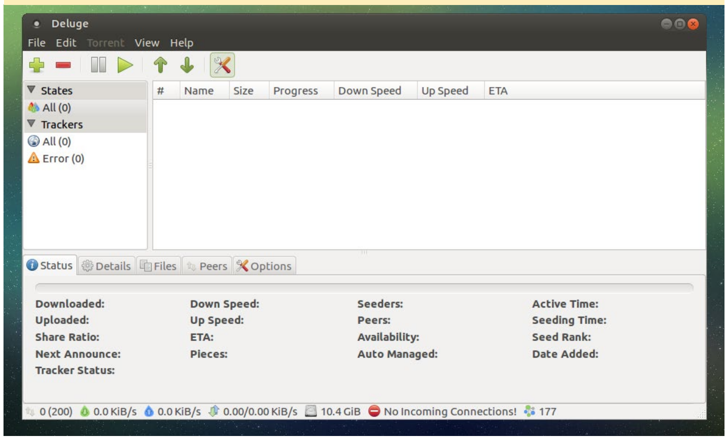
There is a near endless list of BitTorrent clients available for Linux, and everyone will say that the one they use is the best. However, one name you will repeatedly hear as a favorite is Deluge. Its strong popularity is no surprise, since it is not only lightweight while offering a full feature list, but it also is cross-platform and cross-architecture. Deluge shows off a nice-looking and simple-to-use front end that wraps around a libtorrent base. One of the most useful features for us ODROIDians is the ability to have the torrent daemon running on a server and a connect remotely to it via a web interface. This article showcases some of my favorite Deluge features as well as how to use them.

## Getting Started

To play around with Deluge when writing this article, I used an ODROID-XU4, but this guide should work for any other ODROID board. Installation is simple and easy, and can be quickly done with using the Ubuntu Package Manager with the following command:

```
$ sudo apt-get install deluge
```

Figure 1 - The Deluge desktop client


Deluge
File Edit Torrent View Help
States
All (0)
Trackers
All (0)
Error (0)
# Name Size Progress Down Speed Up Speed ETA
Status Details Files Peers Options
Downloaded: Uploaded: Share Ratio: Next Announce: Tracker Status:
Down Speed: Up Speed: ETA: Pieces:
Seeders: Peers: Availability: Auto Managed:
Active Time: Seeding Time: Seed Rank: Date Added:
0 (200) 0.0 KiB/s 0.0 KiB/s 0.00/0.00 KiB/s 10.4 GiB No Incoming Connections! 177

Once installed, we can take a quick look at running and using Deluge on a local machine. This means that the program will be interacting with and running Deluge on the same computer. Launching the program will start both the Deluge daemon and the graphical front end.

## Deluge on a Server

Having Deluge running on a server is a noteworthy feature that deserves its very own section. Deluge is one of the few torrent clients that has the ability to run as a daemon and be interacted with via a web interface. There are actually two daemons that need to be running for this to happen: deluged, and deluge-web. Both daemons may not be typically installed with the basic Deluge pack, but they can be installed with the following commands:

```
$ sudo apt-get install deluge-webui
$ sudo apt-get install deluged
```

Once installed, we need to set up the sysmd scripts for deluged and deluge-web. I would recommend create a more limited user and group to be in charge of running the Deluge services.

```
$ sudo adduser --system \
  --gecos "Deluge Service" \
  --disabled-password --group \
  --home /var/lib/deluge deluge
```

This will create a new group and a new user named "deluge". It is possible to run this with the default "odroid" user, by simply modifying the following service files to use the odroid as a user. Once again, I am going to iterate that for security reasons this is not the best idea. Next we need to create two service files, one for each of the services. Create a file called /

etc/systemd/system/deluged.service and fill it with the following content:

```
[Unit]
Description=Deluge Bittorrent
Client Daemon
After=network-online.target

[Service]
Type=simple
User=deluge
Group=deluge
#007 full access to the user and
members of the group.
#022 full access to the user run-
ning, and read access to others.
#000 full access to all accounts.
UMask=007

ExecStart=/usr/bin/deluged -d

Restart=on-failure

# Configures the time to wait
before service is stopped force-
fully.
TimeoutStopSec=300

[Install]
WantedBy=multi-user.target
```

Next, we need to create a file for web-ui called /etc/systemd/system/deluge-web.service. The content for the file is shown below:

```
[Unit]
Description=Deluge Bittorrent
Client Web Interface
After=network-online.target

[Service]
Type=simple

User=deluge
Group=deluge
UMask=027

ExecStart=/usr/bin/deluge-web

Restart=on-failure
```

```
[Install]
WantedBy=multi-user.target
```

Now the service can be setup to run on boot and started.

```
$ systemctl enable /etc/systemd/
system/deluged.service
$ systemctl start deluged
$ systemctl status deluged

$ systemctl enable /etc/systemd/
system/deluge-web.service
$ systemctl start deluge-web
$ systemctl status deluge-web
```

With both services running, we can now try to connect to it from a web browser. The web interface is by default waiting on port 8112. This means once everything is up and running, we can connect to it from from a web browser by typing in http://<server-IP>:8112.

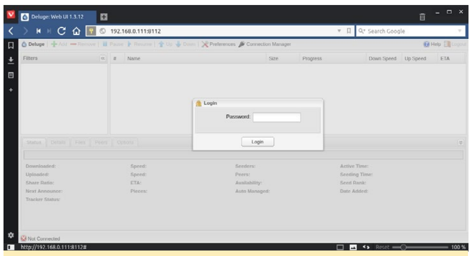


Figure 2 - Web User Interface login

The first time logging in to the system, you will be shown a popup for a password. The default password is “deluge”, followed by the option to change it. After updating the password, you now have full control over your Deluge client.

## Plugins

Beyond the built-in features already available, Deluge has a solid list of additional plugins that can be installed as well. Third party plugins such as batch renamers, and more advanced schedulers can be downloaded from http://bit.ly/2igFfBC.

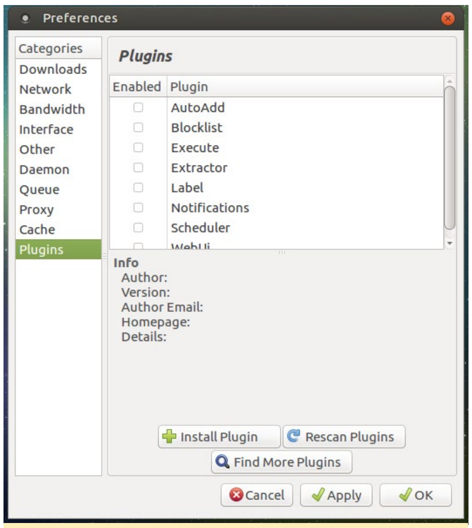


Figure 3 - Plugin menu

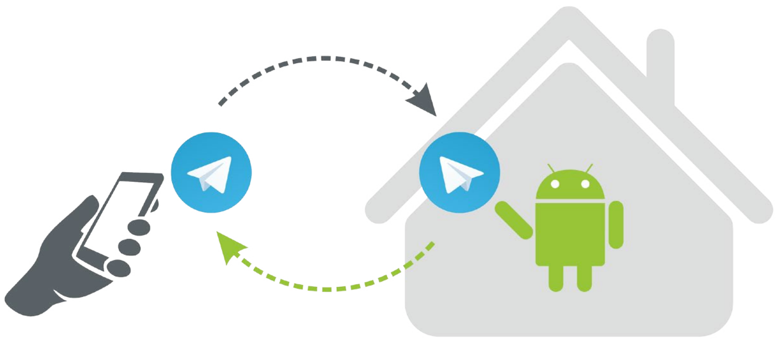
## Conclusion

Deluge is a great open source torrent client that has a plethora of useful features. If you have an extra ODROID or you’re looking for a good way to zest up your current ODROID server, Deluge’s web user interface is a great way to make a quick and easy seedbox.

# TELEGRAM CHATBOT

## ADVANCED HOME AUTOMATION

by Max Volkov



Using an instant messaging chat bot for the user-interface of a home automation project, in some cases, has its advantages over using a web server. For example, our automated house has access to the Internet via a 3G modem. If we intend to use an externally visible web server, we would need a public IP address. This may not always be possible, and we may incur additional expense. One would not face these issues when using a messaging service like Telegram Chatbot.

Telegram is a messaging app that has been gaining popularity, thanks to its distinctive qualities such as speed, reliability, privacy and flexibility. However, what we are most interested in is another feature: an open API for creating chatbots.

Apart from the fact that chatbots are officially supported in Telegram, there is another reason to use this messenger. You can create a custom virtual keyboard that provides unprecedented convenience for the user interacting with the bot. Rather than manually entering a command, the user can tap on a button in the Telegram client app.

In this tutorial, we will learn how to setup and use Telegram chatbot in Linux for a very simple example application of mimicking a smart home. This app is able to light up two LEDs: green and red, according to the input from the user.

## Initial setup

Assume you have Python 2.7 and pip (Python Package Index) already installed on your ODROID device. You also need to install a Python package used in this app - pyTelegramBotAPI, which is a Python implementation for the Telegram Bot API.

```
$ pip install pyTelegramBotAPI
```

Now you need to create a Telegram account if you have not got one yet. Install the Telegram client app on your smartphone, or download it from telegram.org, and install it on your PC. It is straightforward to create an account in the client application.

Next, you need to create your chatbot. It is done using BotFather, which is a bot for creating bots. To do that in the client app, search for BotFather. After finding that contact, click the Start link below it. You'll receive a long message describing all available commands to manage bots. All commands start with a slash ("/"). Use the /newbot command to create a new bot. The BotFather will ask you for a name and username, and after providing that information, an authorization token for your new bot is created. The name of your bot is displayed in contact details and elsewhere.

The Username is a short name, to be used in mentions. Usernames are 5-32 characters long and are case insensitive, but may only include Latin characters, numbers and underscores. Your bot's username must end in "bot", e.g. "tetris_bot" or "TetrisBot".

The token is a string that is required to authorize the bot and send requests to the Bot API. Once you get the token for your bot, save it somewhere to use in your app. In the Telegram client app, search for the Username you had specified earlier and add it to the contact list. Details of managing Telegram bots are described at http://bit.ly/2hlOyQK.

## Run the Python script

Now that you have installed the Telegram client app and created a Telegram user account and a bot, you are ready to run the Python bot app on your ODROID SBC and test it. Save the code snippets listed below into a plain text file and name it, for instance, "chatbot.py". Run the script using the command:

```
$ python chatbot.py
```

Shown below is Python script including some explanations. First, declare the external Python modules used in the script:

```
import telebot
```

```
from telebot import types
```

To create a Telegram bot object, you must submit your bot's token that was mentioned above in the initial setup section.

```
bot = telebot.TeleBot("my_Telegram_bot_token_here")
our_chat_id=0
```

The token our_chat_id is a global variable holding the chat identifier. The meaning of that is explained below in the send_welcome function description.

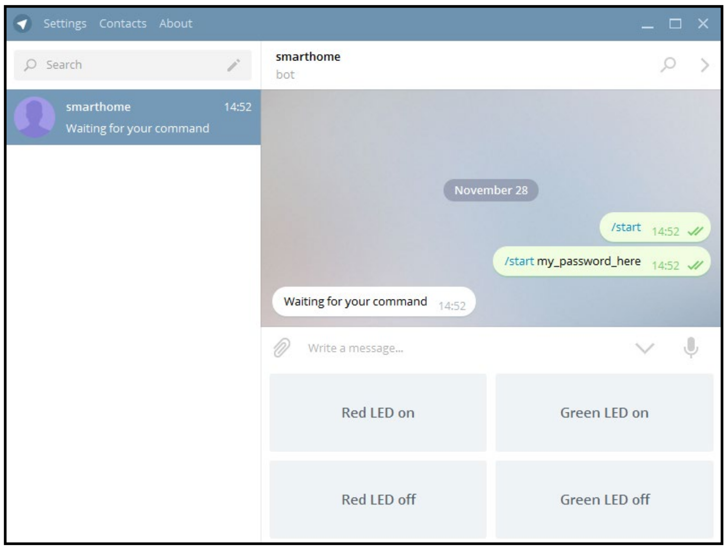
```
def extract_unique_code(text):
    # Extracts the unique_code from the sent /start 
command.
    return text.split()[1] if len(text.split()) > 1 
else None

@bot.message_handler(commands=['start'])
def send_welcome(message):
    global our_chat_id
    unique_code = extract_unique_code(message.text)
    if unique_code: # if the '/start' command con-
tains a unique_code
        if unique_code=="my_password_here":
            our_chat_id=message.chat.id
            send_LED_ctrl_keyboard(our_chat_id)
        else:
            reply = "Sorry, don't know who are 
you..."

def send_LED_ctrl_keyboard(chat_id):
    markup = types.ReplyKeyboardMarkup(row_width=2)
    itembtn1 = types.KeyboardButton('Red LED on')
    itembtn2 = types.KeyboardButton('Green LED on')
    itembtn3 = types.KeyboardButton('Red LED off')
    itembtn4 = types.KeyboardButton('Green LED off')
    markup.add(itembtn1, itembtn2, itembtn3, itemb-
tn4)
    bot.send_message(our_chat_id, "Waiting for your 
command", reply_markup=markup)
```

We have defined three functions above. The first one is just to extract the user password from the /start command line. According to the official Telegram bot guide, all bot developers are required to support a few basic commands: /start, /help, /settings.

The /start command begins an interaction with the user, such as sending a greeting message. In our script, the /start command carries the payload, which means that after the /start keyword, you must specify your user password so that the script could identify you.

This is done in the beginning of the second function send_welcome, which is called when the chatbot receives /start command. If the user password specified after /start keyword is identical to the one in the string value specified instead of "my_password_here" in the script, then the function send_welcome does two things:

1. saves chat id of the current message to global variable our_chat_id that will be used later in the script;
2. creates a custom keyboard with 4 buttons and displays it to the authorized user along with the message "Waiting for your command" by calling the function send_LED_ctrl_keyboard.



On the contrary, if the password does not match, the function sends a response message indicating that the user is not known. Let us examine what chat id is, as well as making a virtual keyboard in more detail.

Chat id is a numerical value that Telegram messaging service assigned to the communication session with the chatbot. Theoretically, anyone from Telegram can contact our bot. Therefore, it makes sense to use it as the identifier of the session, in which the user specified correct password. This chat id is stored in the global variable our_chat_id, and will be used in further communications. Moreover, this chat id can be stored in a file or in a database, and used in the next sessions. In this case, there is no need to enter a password every time the script is run.

In the send_LED_ctrl_keyboard function, we make a custom virtual keyboard with 4 buttons for the purpose of convenience. Tapping on a button has the same effect as typing text on a keyboard and sending it to the bot.

A button is defined by calling the types.KeyboardButton method. An argument is a text string that will be displayed on the button and sent to the chatbot by tapping on that button.

# CHROME DEATH

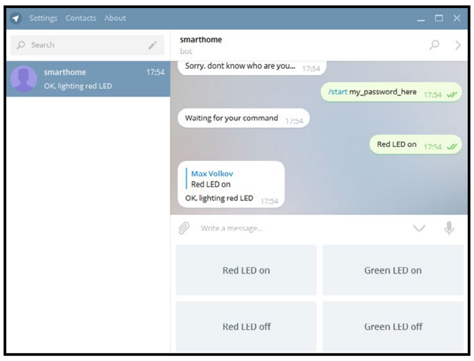
## A CYBERPUNK-THEMED ACTION GAME THAT WILL KEEP YOUR ADRENALINE PUMPING

by Bruno Doiche

Survive in an endless racer set on a cyberpunk-esque city, where you keep running for your life. With agents blocking your way, your reflexes are your tool to keep going for the longest time possible. Based on that cool feel from the 1980s VHS movies and with a soundtrack that is a show in itself, Chrome Death is a thriller that will make your Android gaming collection a little brighter. With easter eggs hidden here and there, you will certainly lose a couple of lives enjoying this fast paced game!

https://play.google.com/store/apps/details?id=com.newmark.chromedeathandroid&hl=en

A game with sublte challenges, this one will test your reflexes for sure



smarthome 17:54
OK, lighting red LED

smarthome
bot

Sorry. dont know who are you... 17:54

/start my_password_here 17:54

Waiting for your command 17:54

Red LED on 17:54

Max Volkov
Red LED on
OK, lighting red LED 17:54

Another option to perform the same action is just to type a message, e.g., “Green LED off” on your keyboard and click “send”. But why bother typing if there is a button?

```
@bot.message_handler(func=lambda
message: True)
def echo_all(message):
    global our_chat_id
    if message.chat.id==our_chat_
id:
        if message.text == ‘Red
LED on’:
            bot.reply_to(message,
‘OK, lighting red LED’)
            return
        if message.text == ‘Green
LED on’:
            bot.reply_to(message,
‘OK, lighting green LED’)
            return
        if message.text == ‘Red
LED off’:
            bot.reply_to(message,
‘OK, putting out red LED’)
            return
        if message.text == ‘Green
LED off’:
            bot.reply_to(message,
‘OK, putting out green LED’)
            return
        bot.reply_to(message,
‘Not understood’)
    else:
        bot.reply_to(message,
“Sorry, don’t know who are you…”)
```

The function echo_all is a message handler, and is called every time our chat bot receives a message except for /start command, that is handled by send_welcome function. First, the incoming message’s chat id is checked. If it’s “our” chat (with an authorized user) then it continues, otherwise it displays an error message and exits. If the check passes, then we analyze the contents of message.text property, whose name speaks for itself. If it is recognized as one of the valid commands, the appropriate action is done. In the echo_all function above, it is only limited by the response message to the user. In the real smart home project, any action can be carried out, such as sending a command to the lower level, or activating a GPIO output pin. A reply of misunderstanding is sent in that rare case when an authorized user tried to type a command manually instead of using a button, and there’s an error in that “hand-written” command.

```
bot.polling()
```

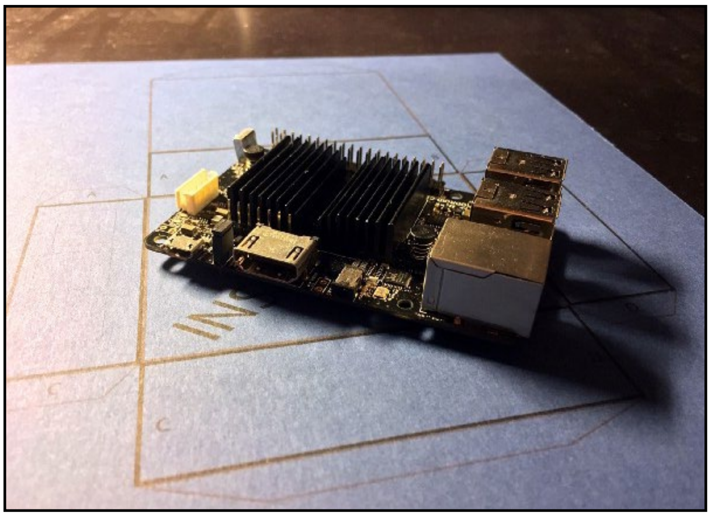
Last, but not least, this line of code is the call blocking method called bot.polling that makes our chatbot waiting forever for an incoming message.

## Notes

In this article, we discussed a very simple version of the Telegram chatbot application. It can be enhanced further by adding a variety of options. For example, consider creating a multilevel “keyboard system” with a different keyboard configuration for each branch of each level. Also, it is possible to attach a keyboard to a message, referred to as the inline keyboard. This and other features are described in detail in the relevant section of the Telegram website at http://telegram.org.

# ODROID-C1/C2 PAPER CASE

by @thekillercarrot



Not a long ago, I bought a wonderful board called the ODROID-C2. Although it has a smaller community of my already retired Raspberry Pi, it is a product of the highest quality and reliability. I decided to start installing all of the things that I needed to make it fully functional. When I had all of the software perfectly configured, there was only one thing missing: a case. Since I do not yet have a 3D printer, I decided to put my hands to work and make my own paper case.

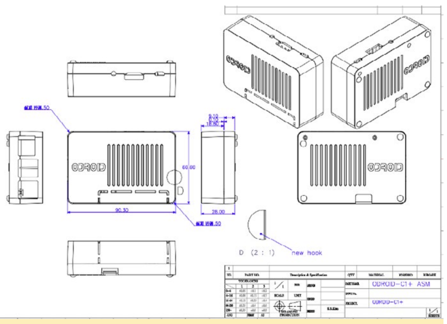


Figure 1 - The measurements for the paper case

After some time searching the web for the measurements of the ODROID-C2, I downloaded random 3D modeling files from other ODROID-C2 cases in order to use their measurements. I thank the person that made the file show in Figure 1, because it was the only file where I could find some numbers for use with my paper case.

After some hours with Photoshop, I was able to say that I had a pretty decent case, which is shown in Figures 2 and 3. I know that it's not a Picasso, but it's a cool case, and you can put some stickers on it!

I recommend printing the PDF file on the thickest piece of cardboard that your printer can handle. Alternatively, you can print it on plain paper and glue that to a piece of thick cardboard.

To make your own paper case, download the file from http://bit.ly/2ifvFyG for free. If you have comments, questions and suggestions, please visit the original article at http://bit.ly/2ipdVhu.

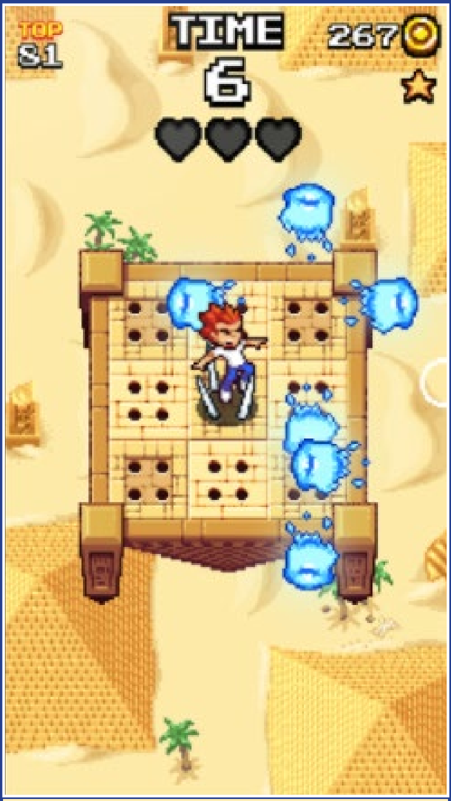
# PIXEL DODGERS

## FASTEN YOUR FINGERS AND DODGE FIREBALLS

by Bruno Doiche

Act fast to dodge as many pixel blasts as you can before being blown to smithereens! In Pixel Dodgers, you need to keep track of where you are and where you need to go in order to escape the pesky fireballs. Simple and intuitive gameplay will keep you dodging all day. The best part is using it with a keyboard and feel like you are playing some sort of old MSX-style game!

https://play.google.com/store/apps/details?id=com.bigbluebubble.pixeldodgers



Dodging fireballs on a flying platform on ancient Egypt with traps on the floor? Count me in!

# ODROID-C2 MANUAL

## A GUIDE FOR ALL EXPERTISE LEVELS

**by Rob Roy (@robroy)**

**ODROID-C2 User Manual cover**

The official user manual for the ODROID-C2 was recently released on the ODROID Magazine website, and is available for direct download at http://bit.ly/2hM1FH6, via the forums at http://bit.ly/2i5F7nM, and on the Google Play Store at http://bit.ly/2iCuupA.

The ODROID-C2 is one of the most powerful low-cost 64-bit Single Board Computers available, as well as being an extremely versatile device. Featuring a fast, quad-core AmLogic processor, advanced Mali GPU, and Gigabit Ethernet, it can function as a home theater set-top box, a general purpose computer for web browsing, gaming and socializing, a compact tool for college or office work, a prototyping device for hardware tinkering, a controller for home automation, a workstation for software development, and much more.

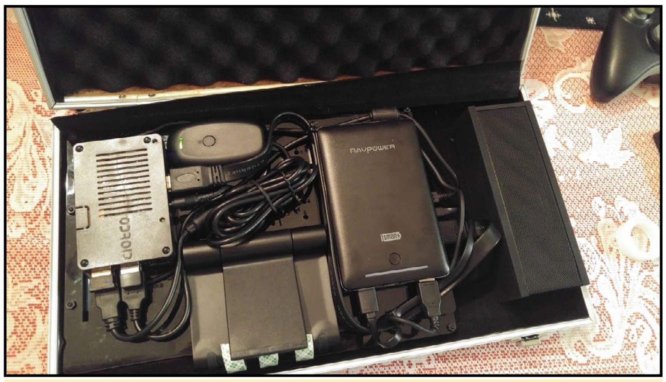
Some of the modern operating systems that run on the ODROID-C2 are Ubuntu, Android, and ARCH Linux, with thousands of free open-source software packages available. The ODROID-C2 is an ARM device, which is the most widely used architecture for mobile devices and embedded computing. The ARM processor's small size, reduced complexity and low power consumption makes it very suitable for miniaturized devices such as wearables and embedded controllers.

# PORTABLE ARCADE STATION

by @LtBenjamin

I created a portable arcade station with a 10" touchscreen using the electronics listed below. The unit plays Atari, Super Nintendo, Nintendo 64 and Nintendo DS games perfectly. Dreamcast and PS games very well with adjustments, and it can run a few PSP games decently.

It runs for almost 7 hours on one charge, and takes 10-12 hours to recharge with a 2.5A charger. For comments, questions, and suggestions, please visit the original thread at `http://bit.ly/2ia2eh7`.


Everything is very neatly arranged inside the poker chip case

The touchscreen fits on top of the rest of the components

The case is portable and looks very James Bond

The Xbox 360 controllers work well with all of the emulators

## The arcade station setup

ODROID-C2 with case
Bluetooth module 2
WiFi Module 0
Dknight Magicbox 2 Bluetooth speaker and microphone
RAVpower 16750 power bank - 4.5A total - 5V/2.4A and 5V/2.1A ports
Waveshare 10.1 inch touchscreen 1280x800 (5V 2.5A)
16GB eMMC (Android)
16GB 48mb/sec Samsung microSDHC card
Xbox 360 wireless receiver
Anker multi angle stand
Poker chip set case

Turok runs very well on the portable arcade station

# ODROIDS AROUND THE WORLD

## THE INTERNATIONAL REACH OF HARDKERNEL'S POPULAR SINGLE BOARD COMPUTERS

by Rob Roy (@robroy)

Hardkernel introduced its first ODROID computer in 2009 (http://bit.ly/1Gx5Lr1), and has since become a leader in single board computers with the recent introduction of the ODROID-C2 and ODROID-XU4. They continually release cutting-edge development boards, and their catalog is extensive:

### Current products

ODROID-C2
ODROID-XU4
ODROID-C1+
ODROID-C0

### Displays

ODROID-VU8C
ODROID-VU5
ODROID-VU7 Plus
ODROID-VU7
3.5inch Touchscreen Sh
C1 3.2inch TFT+Touchsc
16x2 LCD + IO Shield
LED Matrix Shield
ODROID-SHOW2

### Development Kits

C Tinkering Kit
USB-UART Module Kit
Xprotolab Plain

### Add-on Boards

CloudShell for XU4
Expansion Board
USB IO Board
XU4 Shifter Shield
Universal Motion Joypad
USB3.0 to SATA Bridge
U3 IO Shield
U3 Shield Tinkering Ki

### Sensors

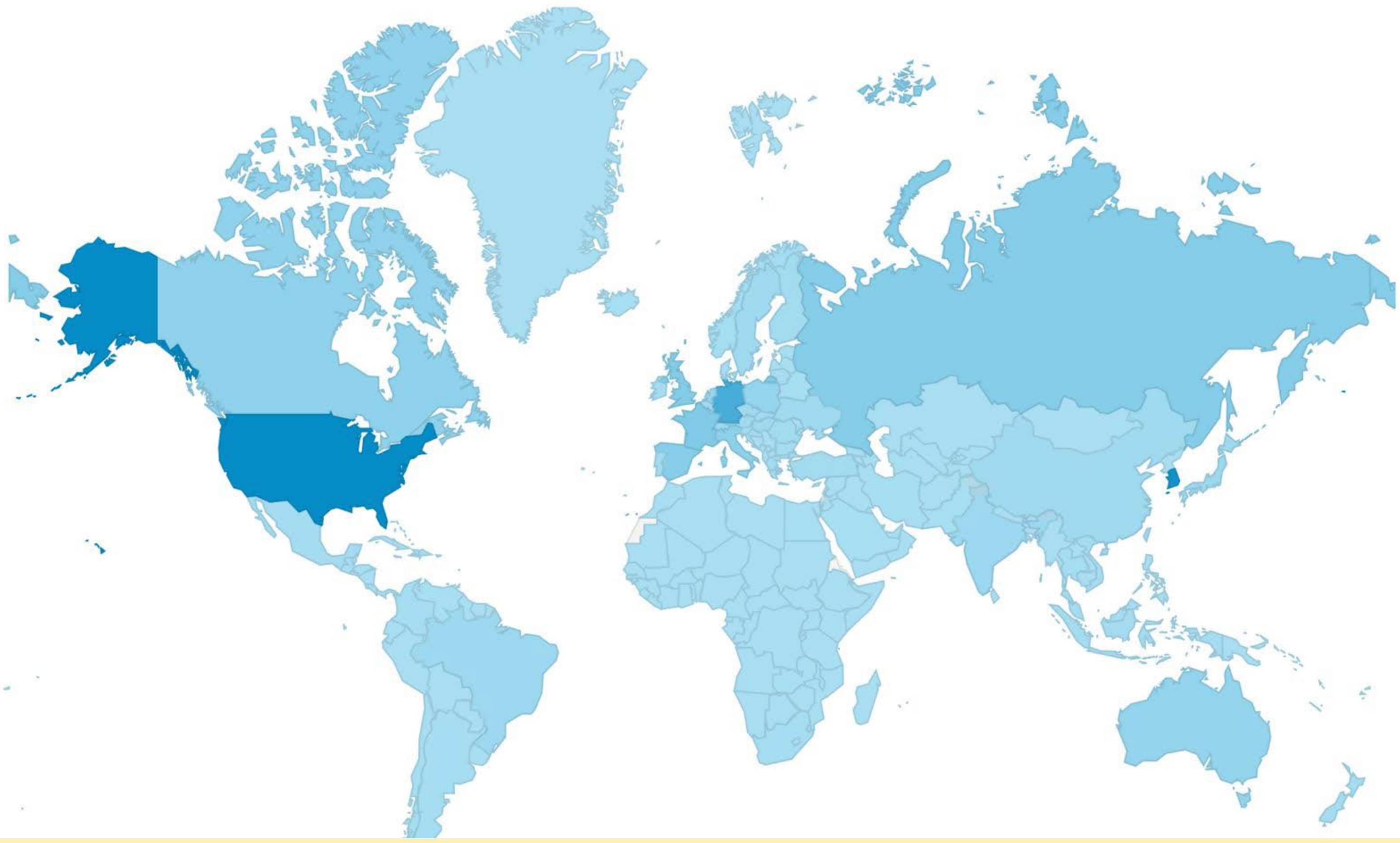
myAHRS+
Weather Board 2

### Legacy products

ODROID
ODROID-7
ODROID-A4
ODROID-PC
ODROID-S
ODROID-T
ODROID-A
ODROID-U3
ODROID-U2
ODROID-X2
ODROID-E7
ODROID-Q2
ODROID-XU3 (+Lite)
ODROID-XU (+Lite)
ODROID-X
ODROID-C1
ODROID-Q
ODROID-XU+E
Smart Power
HiFi Shield for C2/C1+
ODROID-UPS
ODUINO One
UPS2 for U3
Weather Board
ODROID-VU
ODROID-Show
ODROID-W

News of the usefulness and affordability of ODROID boards has spread around the world. Hardkernel has received orders from nearly 150 countries:

Albania
Algeria
Andorra
Angola
Argentina
Armenia
Aruba
Australia
Austria
Azerbaijan
Bahrain
Bangladesh
Barbados
Belarus
Belgium
Bermuda
Bolivia
Bosnia-Herzegovina
Brazil
British Virgin Islands
Bulgaria
Burkina Faso
Cambodia
Canada
Cape Verde
Chile
China
Colombia



**Hardkernel has sold its single board computers in almost 150 countries, and has visitors to its website in 200 countries!**

Costa Rica
Croatia
Curacao
Cyprus
Czech Republic
Denmark
Dominican Republic
Ecuador
Egypt
El Salvador
Estonia
Faroe Islands
Finland
France
French Guiana
French Polynesia
Gambia
Georgia
Germany
Ghana
Gibraltar
Greece
Greenland
Guadeloupe
Guatemala
Honduras

Hong Kong
Hungary
Iceland
India
Indonesia
Iran
Ireland
Isle of Man
Israel
Italy
Ivory Coast
Jamaica
Japan
Jordan
Kazakhstan
Kenya
Kosovo
Kuwait
Kyrgyzstan
Laos
Latvia
Lebanon
Libya
Liechtenstein
Lithuania
Luxembourg

Macau
Macedonia
Madagascar
Malaysia
Maldives
Mali
Malta
Marshall Islands
Martinique
Mauritius
Mexico
Moldova
Mongolia
Montenegro
Morocco
Namibia
Nepal
Netherlands
Netherlands Antilles
New Caledonia
New Zealand
Nicaragua
Nigeria
Norway
Oman
Pakistan

Palau
Panama
Paraguay
Peru
Philippines
Poland
Portugal
Puerto Rico
Qatar
Reunion
Romania
Russia
Saudi Arabia
Senegal
Serbia
Singapore
Slovakia
Slovenia
South Africa
South Korea
Spain
Sri Lanka
Sweden
Switzerland
Syria
Taiwan

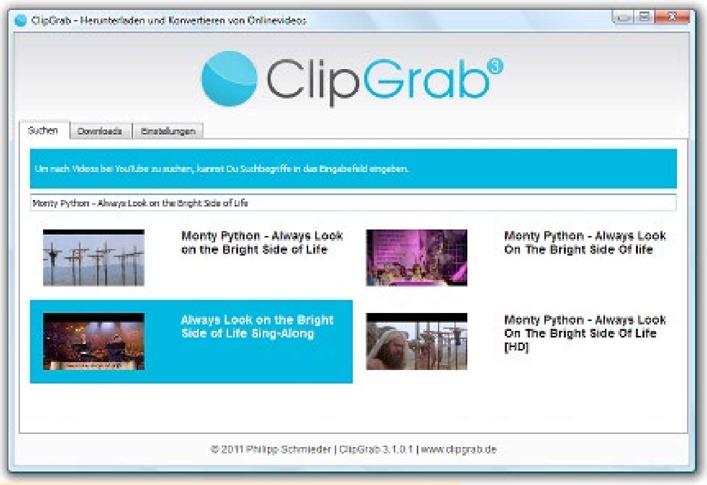
Tajikistan
Tanzania
Thailand
Tunisia
Turkey
U.S. Virgin Islands
U.S.A
Uganda
Ukraine
UAE
United Kingdom
Uruguay
Uzbekistan
Venezuela
Vietnam
Zimbabwe

# CLIPGRAB

## DOWNLOAD YOUR FAVORITE VIDEOS FOR OFFLINE VIEWING

by Tobias Schaaf (@meveric)

ClibGrab is a program that allows to download and convert Videos from sites as YouTube, DailyMotion and other sites. You can choose in which format it should be converted, such as MPEG4, MP3 or OGG. It uses ffmpeg to convert the files.


ClipGrab - Herunterladen und Konvertieren von Onlinevideos

Um nach Videos bei YouTube zu suchen, kannst Du Suchbegriffe in das Eingabefeld eingeben.

Monty Python - Always Look on the Bright Side of Life

Monty Python - Always Look on the Bright Side of Life

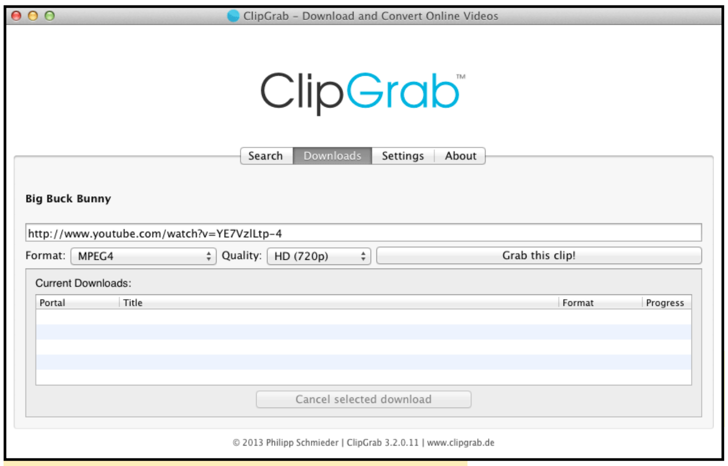
Monty Python - Always Look On The Bright Side Of life

Always Look on the Bright Side of Life Sing-Along

Monty Python - Always Look On The Bright Side Of Life [HD]

© 2011 Philipp Schmieder | ClipGrab 3.1.0.1 | www.clipgrab.de

Browsing the available files


ClipGrab – Download and Convert Online Videos

Big Buck Bunny

http://www.youtube.com/watch?v=YE7VzlLtp-4

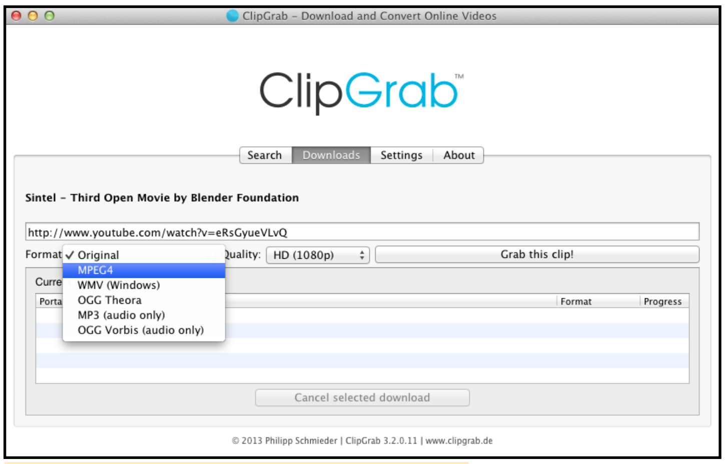
Format: MPEG4 Quality: HD (720p)

Current Downloads:

Portal | Title | Format | Progress

© 2013 Philipp Schmieder | ClipGrab 3.2.0.11 | www.clipgrab.de

Selecting a file to download


ClipGrab – Download and Convert Online Videos

Sintel – Third Open Movie by Blender Foundation

http://www.youtube.com/watch?v=eRsGyueVLvQ

Format ✓ Original Quality: HD (1080p)

MPEG4
WMV (Windows)
OGG Theora
MP3 (audio only)
OGG Vorbis (audio only)

Portal | Format | Progress

© 2013 Philipp Schmieder | ClipGrab 3.2.0.11 | www.clipgrab.de

Configuring the download options

### Supported sites

YouTube
Vimeo
Dailymotion
metacafe.com
youku.com
myspass.de
myvideo.de
clipfish.de
collegehumor.com
Other sites should be working as well.

## Installation

You can download the program from my repository, which is detailed in one of my previous ODROID Magazine articles at http://bit.ly/2icmAUQ. The armhf version can be downloaded from the all/main package list and the arm64 version from the jessie/main package list with the following command:

```
$ apt-get install clipgrab
```

For comments, questions and suggestions, please visit the original thread at http://bit.ly/2hv9Awo.

# KODI SCREENSAVER

## CONTROL YOUR CEC-COMPATIBLE TV MONITOR WITH THIS SMOOTH FEATURE

by @rooted

In this article, I will show you how to turn off your TV when the Kodi screensaver activates, and turn the TV on when the Kodi screensaver is deactivated. Before you begin, make sure that your TV has functional CEC capabilities.

## Installation

First, navigate to Kodi Settings -> Add-ons -> Install from repository -> Kodi Add-on repository -> Services, and install the "Kodi Callbacks" add-on.

Next, mount the / and /system partitions as read/write using a Terminal window:

```
$ su
$ mount -o remount,rw /
$ mount -o remount,rw /system
```

Add the following line to the file /init.odroidc2.rc:

```
chmod 666 /sys/class/amhdmitx/amhdmitx0/cec
```

Then, download the files "cecon" and "cecoff" from https://db.tt/ai1DNnFh, copy them to /bin and make them executable:

```
chmod +x /bin/cecoff
chmod +x /bin/cecon
```

Open Kodi and navigate to Kodi Settings -> Add-ons -> My add-ons -> Services -> Kodi Callbacks. Add the following two tasks under the "Tasks" tab:

```
Task 1
Task = script
Script executable file = /bin/cecoff

Task 2
Task = script
Script executable file = /bin/cecon
```

Under the "Events" tab, add the following two events:

```
Event 1 -> Choose event type -> on Screensaver Activated
Task = Task 1

Event 2 -> Choose event type -> on Screensaver Deactivated
Task = Task 2
```

Finally, reboot the system. Your TV will now turn off when the screensaver is activated. You can adjust the timeout by adjusting the screensaver time. If you are using an IR Remote such as the Hardkernel remote, you can press a button on the remote to wake up the TV or simply turn the TV back using the TV remote. For comments, questions and suggestions, please visit the original thread at `http://bit.ly/2i5OWSz`.

# REAR VIEW CAMERA

## STAYING SAFE ON YOUR BICYCLE

by Brian Kim

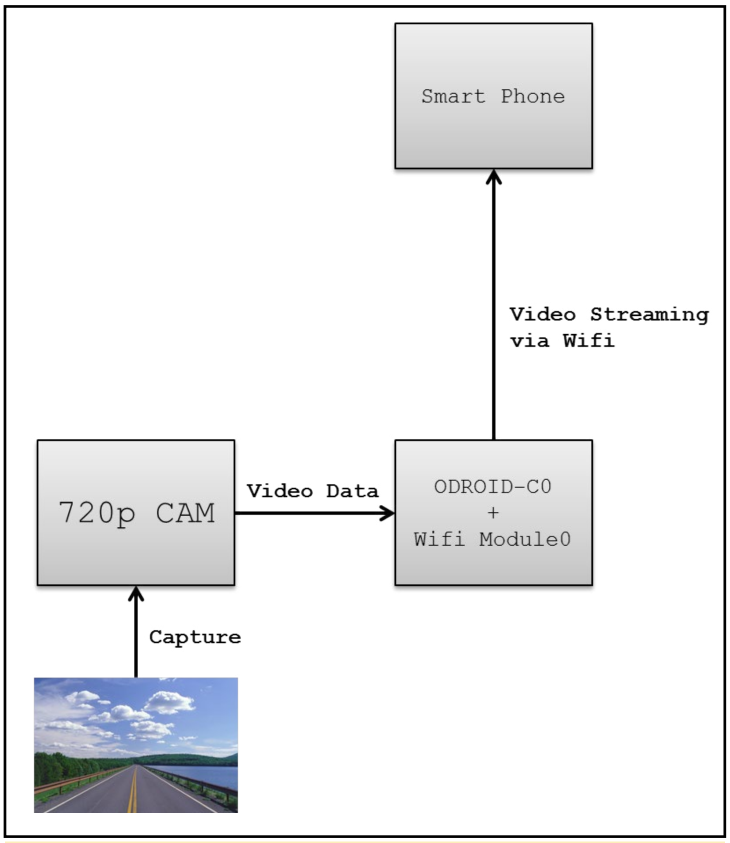
When I moved to my new apartment, one of my friends let me borrow his road bike. A road bike is a type of lightweight bicycle that is designed to go fast speeds on smooth paved roads. One day, I got injured from falling off the bike, since I was new and inexperienced at riding road bicycles. Nevertheless, I can ride faster on a road cycle than any other type of bicycle, and the thrill of speed keeps me riding. Unsurprisingly, one of my favorite hobbies became riding road cycles.

It depends on the laws of each country, but in South Korea, it is legal to ride a bicycle on the roads. However, since I share the roads alongside cars, I need to ride fast in order to not disturb the flow of traffic. It's when I am riding at full speed on the road that riding really becomes exciting for me. Since I riding next to cars and trucks, I am always worried that a car might hit me from behind. It makes me very uneasy that I don't have a view of what is behind me. So, I decided to make a rear view camera system myself using an ODROID-C0.

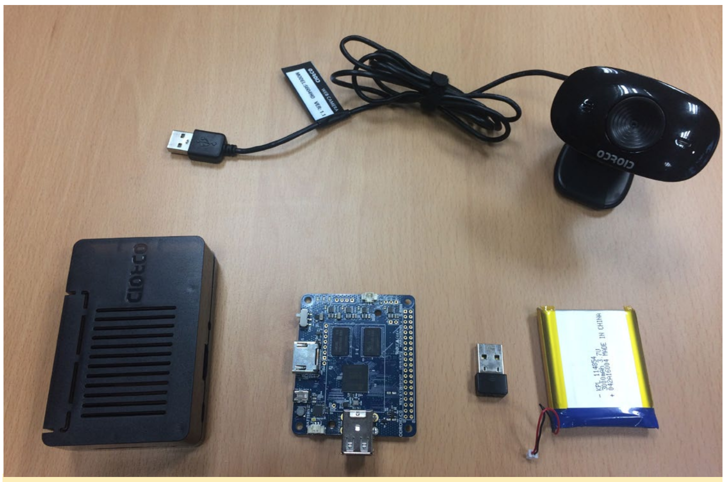
The DIY rear view camera needs high bandwidth to transmit the real time video. I decided it is a good approach to use wifi as the interface for this project, as a wireless connection gives us better spatial flexibility. To avoid making an additional display device, I decided to use my smartphone as the rear view display. The rear view camera would be a Wifi AP (Access Point) that will allow the smartphone to connect to it. For this project the Wifi AP uses a ODROID Wifi module 0 and an ODROID-C0. The overview of the concept is shown in Figure 1. A 720p camera captures the rear view and then the ODROID-C0 encodes the video frame and transmits the encoded video data via Wifi. We can then watch our rear view on our smartphone after connecting to the streaming video server hosted on the ODROID-C0.

These are the required hardware components in order to make a DIY rear view camera:

**ODROID USB-CAM 720p**
**ODROID-C0**
**16GB eMMC Module C1+/C0 Linux (Black)**
**Wifi Module 0**
**ODROID-C2/C1+ Case**



Rear View Camera Concept Overview


Hardware components for the DIY Rear View Camera

**3000mAh Battery**
**USB-A Female Dual Port**

I soldered a USB-A Female Dual Port to the ODROID-C0 as seen in Figure 2 because the ODROID-C0 does not come with the USB ports solder in. There is also an ODROID USB-CAM 720P Camera, 3000mAh Battery, WiFi module 0, ODROID-C0, and ODROID-C2/C1+ case. First, the DIY rear view camera needs to have the software configured before being installed to the bicycle. The software configurations can be divided into two parts: Wifi AP settings and the video streaming server settings.

## WiFi Access Point configuration

Hostapd is a user space daemon for access points and authentication servers. It can be used to create a wireless hotspot using a Linux computer. Some additional packages are needed for hostapd. An explaination about the hostapd setting can be found on the Hardkernel wiki page for hostapd at http://bit.ly/2fJTr4h. Hostapd can be installed with the following commands:

```
$ sudo apt-get update
$ sudo apt-get install libnl-3-dev \
  libnl-genl-3-dev libssl-dev hostapd
```

The smartphone connects to the rear view camera, which corresponds to the “ssid” in the configuration file after setting to AP mode. “wpa_passphrase” is the password need for connecting to the AP. The hostapd configuration files for Wifi module 0 are as follows:

```
/etc/hostapd/hostapd.conf
interface=wlan0
driver=nl80211
ssid=ODROID_REARCAM
hw_mode=g
channel=6
macaddr_acl=0
auth_algs=1
ignore_broadcast_ssid=0
wpa=2
wpa_passphrase=saferiding
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP
rsn_pairwise=CCMP

/etc/default/hostapd
DAEMON_CONF=”/etc/hostapd/hostapd.conf”
DAEMON_OPTS=”-B”
```

The ODROID-C0 must assign the IP address to the connected devices. I used dnsmasq in order to allocate dynamic IP addresses to the connected nodes. Dnsmasq provides network infrastructure for small networks such as: DNS, DHCP, router advertisement, and network boot.

```
$ sudo apt-get install --reinstall dnsmasq
$ mv /etc/dnsmasq.conf /etc/dnsmasq.conf.orig

The conf file for dnsmasq can be configured as shown.
/etc/dnsmasq.conf
domain-needed
bogus-priv
no-resolv
no-poll
server=/example.com/192.168.1.5
server=8.8.8.8
server=4.4.4.4
local=/example.com/
address=/doubleclick.net/127.0.0.1
no-hosts
addn-hosts=/etc/dnsmasq.d/hosts.conf
expand-hosts
domain=example.com
dhcp-range=192.168.1.20,192.168.1.50,72h
dhcp-range=tftp,192.168.1.250,192.168.1.254
dhcp-option=option:router,192.168.1.1
dhcp-option=option:ntp-server,192.168.1.5
dhcp-option=19,0 # ip-forwarding off
dhcp-option=44,192.168.1.5 # set netbios-over-TCP/IP
aka WINS
dhcp-option=45,192.168.1.5 # netbios datagram distri-
bution server
dhcp-option=46,8           # netbios node type
```

The ODROID-C0 must be set to the server, or gateway, IP address of "192.168.1.1". The rear view displayer will connect to this IP address.

```
/etc/network/interfaces
# interfaces(5) file used by ifup(8) and ifdown(8)
# Include files from /etc/network/interfaces.d:
source-directory /etc/network/interfaces.d

auto lo
iface lo inet loopback

auto eth0
iface eth0 inet dhcp

auto wlan0
iface wlan0 inet static
        address 192.168.1.1
        netmask 255.255.255.0
```

## Real-time Camera Streaming Server Configuration

The ffmpeg Linux package is for encoding and streaming video data, so we can use it for our 720p camera stream. Ffserver is able to stream many kinds of video formats but, we need real time video playback to watch what is happening behind us. The ODROID 720p camera supports not only raw video data, but can also encode each video frame as jpeg. Therefore, mjpeg (Multipart JPEG) format is good choice to reduce the video encoding overhead and network bandwidth usage.

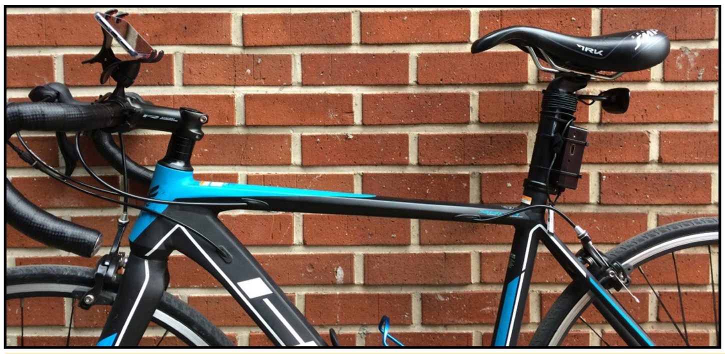
```
$ sudo apt-get install ffmpeg

/etc/ffserver.conf
HTTPPort 8090
HTTPBindAddress 0.0.0.0
MaxHTTPConnections 2000
MaxClients 1000
MaxBandwidth 5000

<Feed cam1.ffm>
File /tmp/cam1.ffm
FileMaxSize 100M
</Feed>

<Stream cam1.mjpg>
Feed cam1.ffm
Format mpjpeg
VideoCodec mjpeg
VideoFrameRate 20
VideoBitRate 4096
VideoSize 640x480
NoAudio
</Stream>

<Stream stat.html>
Format status
ACL allow localhost
ACL allow 192.168.0.0 192.168.255.255
</Stream>
```

To start the streaming server automatically on every boot, ffserver and ffmpeg commands are added to the /etc/rc.local boot script file. Ffmpeg encodes the video data from the 720p camera with the V4L2 (Video for Linux 2) interface, and then send the video frames to the streaming server. Ffserver, which is the video streaming server, receives the encoded video frame and transmits the video frame in mjpeg format to the connected client, which is a smartphone web browser.

```
/etc/rc.local
ffserver -d -f /etc/ffserver.conf&
ffmpeg -f v4l2 -s 640x480 -r 20 -vcodec mjpeg -i /
dev/video0 http://localhost:8090/cam1.ffm

exit 0
```

## Rear View Camera Installation

The software configurations for the DIY rear view camera are done after finishing the Wifi AP configuration and the video streaming server setting. The next step is to install the DIY rear view camera to the bike. Attach the 3000mAh battery to the ODROID-C0 and pack them into the ODROID-C2/C1 case. Then, mount the 720p Camera and the ODROID case to your bicycle. I used a cable tie and smartphone holder to hold the rear view camera to my road bicycle as shown in Figure 3. Plug in the 720p camera and Wifi module 0 to the USB


DIY rear view camera installation to the road cycle

ports of ODROID-C0 before turning on the ODROID-C0.

The smartphone, which is a rear view displayer, must connect to the "ODROID_REARCAM" access point via Wifi using the password "saferiding". By opening the webpage http://192.168.1.1:8080/cam1.mjpg on your smartphone, you can have a digital real time rear view of our surrondings. Ride safe and be happy with your DIY rear view camera system.

Charging the camera at home

# 32-BIT EXECUTABLE ON 64-BIT UBUNTU

## CHRONICLES OF A MAD SCIENTIST

by Bo Lechnowsky (@respectech)

"Blast it!" you say, as you are confronted by an error message on your ODROID-C2 running 64-bit Ubuntu:

```
$ sudo ./r3
sudo: unable to execute ./r3: No
such file or directory
```

You check, and in fact the file does exist in the directory, and it is set as executable! You remember running into this problem a couple of years ago, but you don't remember where you put your note on how you fixed it. You mutter to yourself, "This time, I'll make sure to post it on one of my favorite online forums after I figure out how to fix it again!"

The "file r3" Linux command gives the following output:

```
$ file r3
r3: ELF 32-bit LSB executable,
ARM, EABI5 version 1 (SYSV),
dynamically linked, interpreter /
lib/ld-linux-armhf.so.3, for GNU/
Linux 2.6.27, BuildID[sha1]=96b9
4abdd300ad350ceb0b48b4b0461abd48
1c18, stripped
```

That excites some synaptic responses in a long-dormant neural region in your brain. "Aha, it has something to do with a 32-bit binary running on a 64-bit OS!"

After some trial-and-error, you come up with the following fix:

```
$ sudo dpkg --add-architecture
armhf
$ sudo apt install libc6:armhf \
  libncurses5:armhf
  libstdc++6:armhf
```

After installing those libraries, you attempt to run your executable again. This time, it works! "Victory is mine! Technology must kneel before my greatness!" The evening wears on as you build your digital framework for world domination. It is not long before you try to run a graphical 32-bit executable and you encounter:

```
error while loading shared li-
braries: libX11.so.6: cannot open
shared object file: No such file or
directory
```

"Confound it!" you yell! So you counter the message with the following command:

```
$ sudo apt install libx11-6:armhf
```

The system then returns with this output:

```
error while loading shared
libraries: libXt.so.6
```

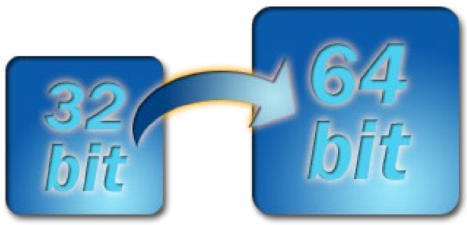
You and the operating system engage in a power struggle:

```
$ sudo apt install libxt6:armhf
error while loading shared li-
braries: libXaw.so.7
$ sudo apt install libxaw7:armhf
error while loading shared li-
braries: libfreetype.so.6
$ sudo apt install
libfreetype6:armhf
```

"We can do this all night!" you say to your digital archrival. For most other applications, this would be sufficient. But for your secret weapon, Rebol 2, you need to run a couple more commands:

```
$ sudo apt install xfonts-100dpi
xfonts-75dpi
$ sudo reboot
```

"Declare your fealty to me, technology!" You put your arms behind your head and feet up on the desk as you bask in the warmth of finally getting to declare lordship over your digital vassals.



# MEET AN ODROIDIAN

## FABIEN THIRIET (@FAB)

edited by Rob Roy (@robroy)

*Please tell us a little about yourself.*

I am 51, and live with my wife Nadine and my 2 sons in Orleans, France, which is 100km south of Paris. For the past 10 years, I have taught network and digital design and maintenance in a French high school. I received an Electrical system degree from the French state for having the right to teach in high school, and I also earned an Engineering degree in computer sciences. Before being a teacher, and after finishing my studies in 1988, I started working with an international company that made Smartcards and chip card readers for banking, health and telecom applications. I developed operating system for these highly secured systems based on Motorola 6805 chips. I also worked on the card readers, designing the PCBs and all the surrounding wiring During the last 3 years with these Smartcard company, I was their Java "evangelist", since SmartCards became more and more "smart", and included a Java Virtual Machine as part of the OS. We placed our JVM in both cards and readers, in order to have very flexible secured solutions. These systems were forward-thinking, because Android from Google has a similar architecture with the powerful Dalvik JVM on top of Linux. That was a very pleasant time, since I travelled around the world a lot, teaching Java for embedded systems, and spent time in the United States, China and South Africa.

After this rich experience, I decided with two other guys from the SmartCard company to make a big jump, and founded our own company in 1999 by developing SMS and MMS solutions for telecom operators. This was when SMS/MMS usage was growing very rapidly, and telecom operators were not really ready to handle hundreds of millions of messages per day.

With my team, we developed powerful and scalable solutions for dealing with SMS/MMS, based on Java and Linux servers, which was mainly on Redhat in those days. This was different from what I did before, since we were handling farms of servers and network clusters, and setting up load balancing machines. The kind of servers we used at this time were Dell PowerEdge 8450 (8 CPUs and 512MB RAM), weighing around 60kg, which were actually less powerful than a single ODROID-C1! In 2005, I decided to sell my shares from the company that I co-founded in order to have more time with my family. Teaching at a local high school close to my home was the appropriate way to accomplish my goal.

With my wife Nadine, we have 2 sons: Antoine, who is 20 years old, and studies medicine in a French University, and Martin, who is 16 years old, and has become a champion of 3D design and printing. He is the one who made the Redtop frame that you may have seen in a previous issue of ODROID magazine. He attends the high school where I am teaching, and took engineering sciences as an additional discipline, and works on his project using an ODROID-C1. My wife is a specialized educator for handicapped. At home, she does plenty of beautiful decorations with forgotten old stuff that she recovers from everywhere.

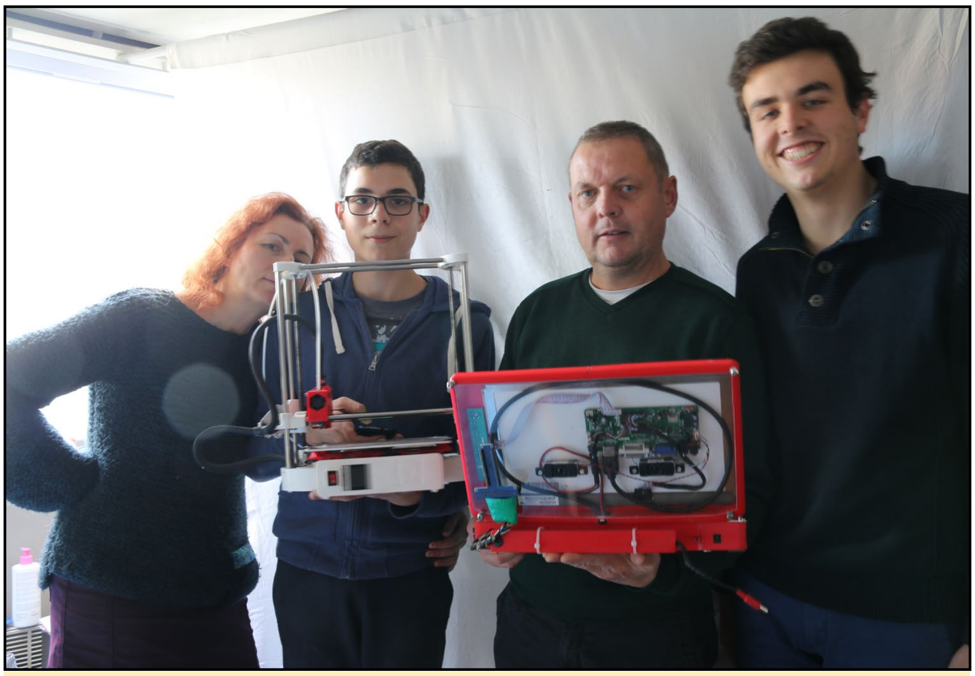
**Fabien with his ODROID dashcam mounted on his car**

*How did you get started with computers?*

I started in the computer world, not with Commodore 64, ZX81, or any such system, but with a Nixdorf mini computer in the early 1980s. Mini computers at this time meant small mainframe systems. Nixdorf was a German company and is now part of Siemens Group.

My parents sold and repaired agricultural machines, and for handling their parts warehouse, they decided to buy a computer. The Nixdorf minicomputer was designed specifically for professional shops, and was the first computer that I worked on. The computer was bundled with software designed for the agricultural machines business, and I did a lot of code modification in Business BASIC, which was the primary programming language for those machines, because it was the early 1980s. The software company working for the Nixdorf local reseller found out that the modifications that I did were quite interesting, and decided to hire me during summer vacations.

An old gem, the Nixdorf mini computer


Fabien and his talented and tech-savvy family showing off their projects

I was really enjoying that new kind of job, and learned a lot, since this minicomputer was able to handle around 10 terminals inside a very primitive network. Terminal addresses on the network were actually set up with DIP microswitches. The Nixdorf minicomputer was able to connect with other remote minicomputers through a 1200 baud modem. Despite this very low rate, it is was not slow, as there were no graphical interfaces at all.

*What attracted you to the ODROID platform?*

In my classroom until 2012, I used an Asus laptop running Ubuntu. In the spring of 2012, the Raspberry Pi foundation released a very unusual tiny computer. I know that ODROID had released a single board computer model before then, but this information had not reached the old European countries yet.

I decided to do a trial with the Raspberry Pi, and replaced all of my Asus laptops with this new toy, mainly for space reasons, since the Pi is very space efficient compared to a traditional laptop. Students loved them at the beginning, but found out that the Pi was very very slow, especially when browsing the Internet.

In 2014, Odroid SBCs finally reach Europe, so I migrated all of my classrooms to the ODROID-C1 platform. This year, I did a second upgrade to the ODROID-C2, which is perfect for what I am doing at school, as it is really fast.

*How do you use your ODROIDs?*

At school, the ODROID is used for everything: web browsing, document writing, Python programming, GNS3 network labs, FreeCad design, video surveillance systems, and robotics. At home, my main computer is the Redtop as I mentioned before, based on an ODROID-C2. I do everything with it, such as preparing my school courses, testing my labs, and building my own systems. As you can see on the different pictures included in this article, I have made many projects with ODROID boards such as:

- A guitar mixer based on Guitarix, handled by an ODROID-XU and Jackd. The XU is wired with a Korg Kontrol2 MIDI device and a Behringer Audio USB adapter.
- An accurate and portable system for marking the geographical boundary of land, based on an ODROID-C1, a 3.2 TFT color touchscreen, and the Hardkernel USB GPS dongle. I developed the software in Python with the ncurses library in order to have a nice semi graphical interface, but no desktop for a lightweight solution.
- A dashcam for my car, in order to record everything in front of my car for insurance purposes, in case of a car accident. The dashcam uses an ODROID-W with a Pi camera.
- A surveillance radar which sends pictures through MMS

Fabien's amazing master project, the RedTop laptop

Fabien is an accomplished musician on electric and acoustic guitar

A futuristic guitar mixer using an ODROID-XU

as soon as something moves in front of dual PIR/microwave sensors. It is made with an ODROID-C1, a Huawei E220 3G USB dongle, and a night vision USB camera along with the dual motion sensors. This system is a ready-to-use solution, and you just need to power the box from a 230V socket.

*Which ODROID is your favorite and why?*

My favorite ODROID is the C2, since the GPIO pin header remains more or less compatible with Raspberry Pi, which is really a big locomotive in the SBC world for add-on boards. The ODROID-XU family is very nice, but unfortunately is missing the Raspberry Pi GPIOs.

*What was your motivation for developing your "RedTop" ODROID laptop?*

My RedTop is really my preferred device. First of all, it was developed with my son, who did a really good job for the CAD part, and second, the RedTop is based on a C2 which is the general purpose computer I am using at school. So, all the stuff that I am doing can be tested and very well prepared with the RedTop in convenient way. The RedTop has a 13-inch color LCD display, so it is very comfortable to work with.

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

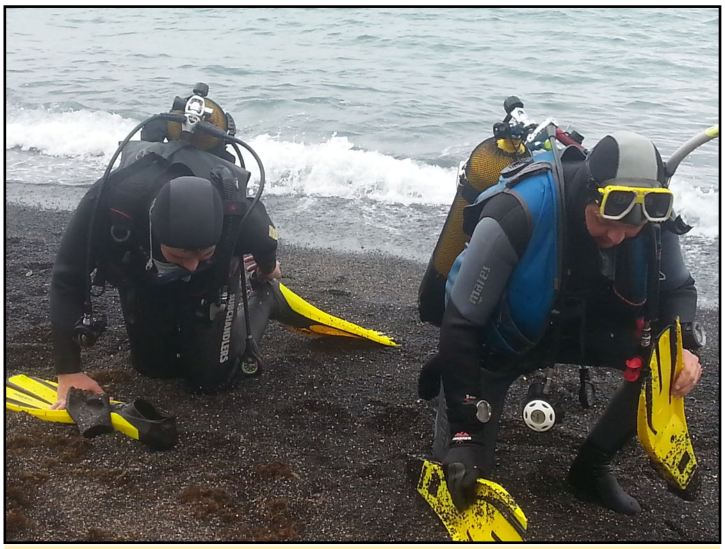
Like most people, I would like to have something faster. A better ARM core with a true 2GHz clock speed would be nice. What is very important, in my opinion, is to remain GPIO-compatible with the Raspberry Pi. Also, SPI and RTC are missing on the C2, and it would nice to have them again, in order to avoid using any add-on boards for that purpose.

*What hobbies and interests do you have apart from computers?*

From April to October, when the Atlantic Ocean temperature becomes pleasant, I practice Scuba diving with my son Martin around the Brittany coast. I am a CMAS 3-star diving instructor. I have dived all around the world in very nice spots like Maldives Islands, Hurghada in Egypt, Cuba, and French Caribbean islands. I also play both acoustic and electric guitar. The last song I played was "Hello" from Adele in a finger-picking style.

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

After more than 30 years, working with digital systems, I can say that learning something, and especially programming, is something you cannot do only at school. Do not just rely on your teachers. Try things by yourself, make errors and find out some solutions alone. The best programmers that I hired during the time that I worked with my own company were people who were curious, with a sense of autonomy while working.


Fabien has practicied Scuba diving all over the world