LineageOS • RemotePI Board • HIFI Shield 2 • Android Development

ODROID Magazine

Year Four
Issue #39
Mar 2017

Create your own ODROID ARCADE Station

The complete tutorial for your setup

- IOT Doorbell: Get a picture of anyone that rings your house
- Home data center: deploy your projects with ease

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

EDITORIAL

Everyone loves games, especially the engineers at Hardkernel. Our featured project this month is a custom project built by Brian, John and Charles to highlight the power of the ODROID platform as a versatile game and console emulator. Using the WiringPi library and the popular ODROID GameStation Turbo gaming image, they designed an Arcade Box that looks awesome and runs your favorite games in 1080p resolution. You can build your own and have a great conversation piece for your next gaming party! Miltos created a useful remote doorbell to see who's at your door by sending an email with a picture, John shows us how to build a home data center with an ODROID-XU4, and Bo continues his Chronicles of a Mad Scientist with an ultrasonic sensor for his vehicle. Tobias presents OpenFodder, a Cannon Fodder clone that captures the beauty of the original game, Nanik discusses analyzing network usage in Android, and Bruno brings us another installment of his Android Gaming series with Causality.

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

Played about 20 games this month, but found only one worth writing about!

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.

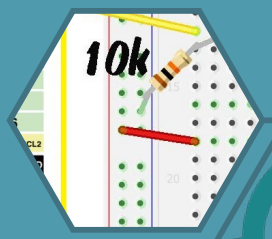
Venkat Bommakanti, Assistant Editor

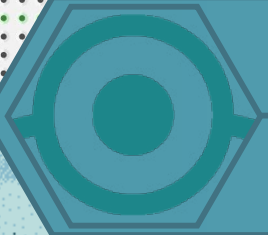
I'm a computer enthusiast from the San Francisco Bay Area in California. I try to incorporate many of my interests into single board computer projects, such as hardware tinkering, metal and woodworking, reusing salvaged materials, software development, and creating audiophile music recordings. I enjoy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

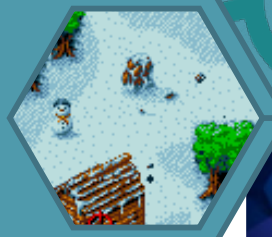
Josh Sherman, Assistant Editor

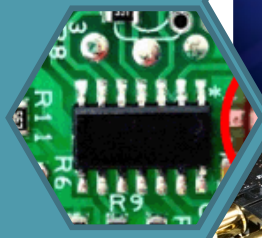
I'm from the New York area, and volunteer my time as a writer and editor for ODROID Magazine. I tinker with computers of all shapes and sizes: tearing apart tablets, turning Raspberry Pis into PlayStations, and experimenting with ODROIDs and other SoCs. I love getting into the nitty gritty in order to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

INDEX









IOT DOORBELL

GET AN EMAIL ALERT OF THE PERSON AT YOUR DOOR

by Miltiadis Melissas (@miltos)

In this project, we'll turn the ODROID-C2 (http://bit.ly/1oTJBya) into a smart IoT doorbell that takes a photo of whoever rings it, and emails that photo to the owner's Gmail account. Additionally, the device will also archive those photos by date and time, giving the owner the ability to check for any suspicious activity or simply to keep records of all the people who ring the bell. It's easy to see how this "smart" doorbell using the ODROID-C2 under the hood is a powerful security and observatory tool, and will be useful for every house.

Hardware requirements

- ODROID-C2 (http://bit.ly/1oTJBya)
- ODROID webcamera (http://bit.ly/2iBHKPD)
- Wi-fi adapter (http://bit.ly/1M4LdiC)
- 1x mini breadboard
- 1x 1 KΩ resistor
- 1x 10 KΩ resistor
- 1x button
- ~8x dumpon wires (the C Tinkering Kit http://bit.ly/1YNPN6k is a great choice for this and future projects)

Software requirements

- 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

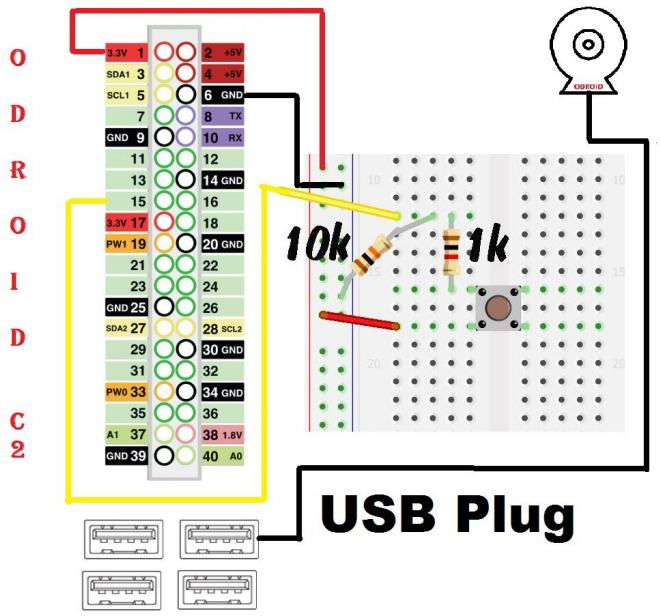


Figure 1 - Doorbell Diagram

and male to male Dupont wires. The female side of the male to female Dupont wire connects to the male header of the ODROID-C2, and the other side connects into the holes of the breadboard. Please refer to Hardkernel's pin layout schematic as you create the connections. The schematic can be found at http://bit.ly/2aXAlmt. Physical pin 1 is VCC and provides 3.3V to our circuit, and we connect the pin on the first vertical line, the one near the edge, of our Breadboard. Since we are going to use pin 6 as the common ground, we connect that to the second vertical line of our breadboard. The rest of the circuit is very simple, and you can follow the diagram as shown in Figure 1.

The ODROID Webcam needs to be plugged into the ODROID-C2 via one of the USB ports available on the board as a final step. The doorbell is controlled through pin 15 as you can see in the figure. Now that the hardware is ready, let's dive

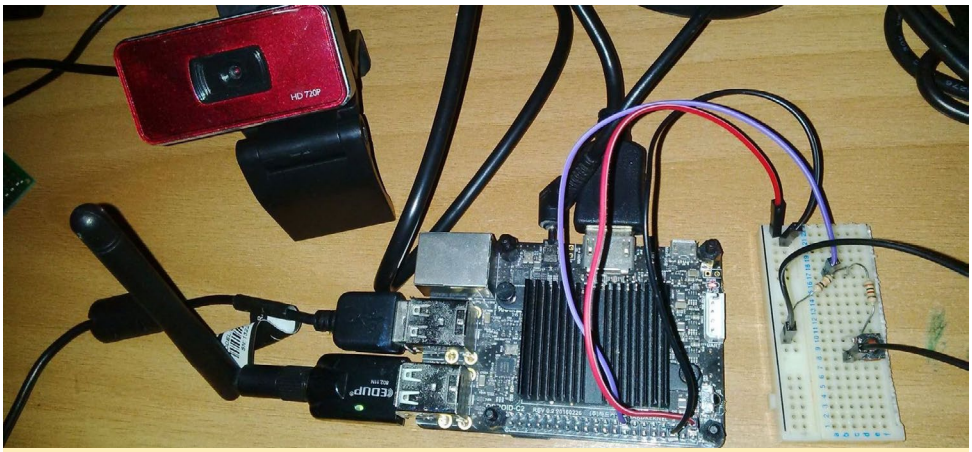


Figure 2 - The C2 complete system

into the code and make that bell ring "intelligently"!

Software

The core of this code was pulled from a Github project available at http://bit.ly/2jEXRbR. Nevertheless, this sample code was heavily modified in order to run properly on the ODROID-C2. Most importantly of all the remapping from RPi.GPIO to WiringPi2 library has been done, since WiringPi2 is the supported by the ODROID-C2. Please refer to the excellent GPIO guide provided by Hardkernel at http://bit.ly/2jEUjWX. All code describe in the following sections in placed between '< … >', with the description written bellow.

The basic odroidbell.py code

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

We start by importing the wiringpi2 library. Instructions on how to install this library for controlling the GPIO pins of the ODROID-C2 can be found at http://bit.ly/2ba6h8o.

```
<import time>
```

We import the time module.

```
<import os>
```

We import the os module.

```
<import glob>
```

We import the glob module.

```
<import sys>
```

We import the sys module.

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

We set up the wiringPi2 module according to the table provide by Hardkernel at http://bit.ly/2aXAlmt.

```
<Button = 3>
```

This is actually the physical pin 15 according to the table at http://bit.ly/2aXAlmt.

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

We set the button as an input.

```
<odroid.
pullUpDnControl(Button,1)>
```

We activate the pull Up and Down resistor. Pull up at this case as the argument 1 denotes that.

```
#loop
```

We are entering a loop...as IoT devices are always at a standby mode.

```
<print("Program Running")>
```

This is just for monitoring purposes.

```
while True:#loops forever till
keyboard interrupt (ctr + C)>
  <if odroid.digitalRead(Button)
== False: #when button is un-
pressed:>
   <sys.stderr.write(".")>
```

If the button of the doorbell is not pressed, we print dots on the screen using the sys library.

```
<time.sleep(1)>
```

We are checking for button press every second.

```
<else:>
  <print("Button Pressed")>
```

The button has been pressed.

```
 #    ------|    photo & Bell
|------ #
    #Get FileName
    <now = time.strftime("Date%m-
%d-%yTime%H-%M-%S")>
```

We declare the variable now with the date and time that we will use on the photo.

```
  #Make command to run odroidC2.
sh
     <command = "bash odroidC2.sh
" +  str(now)>
```

We invoke the odroidC2.sh shell script (see below)

```
  # -- odroidC2.sh is an Shell
script that
    # -- is responsible for tak-
ing the photo and
    # -- making the Doorbell
Noise
```

```
    # --- We insert the "Now" ar-
gument so the python
    # --- script knows what the
filename of the
    # --- picture will be so it
can pass it on into the
    # --- email script (so it
knows what file to email
        #run command
    <os.system(command)>
```

We invoke the system command for running the email script below.

```
  #diagnostics
    <print("Filename:", now)>
```

We print the filename of photo with the present date and time.

```
 # ----| Email      |---- #
    <print("Email")#email>
    <emailcommand = 'sudo python
IoTOdroid.py "This person is at
your door"' + ' "photos/' + now +
'.jpg"'>
```

We email the photo to the owner's Gmail account with the date and time and a subject of "This person is at your door".

```
   <os.system(emailcommand) #run-
ning the Email script with:>
    #-- the subject as "Someone
is ringing the doorbell" and the
filename
    #-- We made before at the
-Photo & Bell- section
```

And finally we send it.

```
 # -- End Diagnostic Info
    print("Done Process")
```

All is done, so end the script

```
   #-space out for next "Press of
Button"
    <print("")>
    <print("")>
```

Make room for the next cycle (next doorbell ring).

OdroidC2.sh shell script

The OdroidC2.sh script is responsible for taking the photo and making the doorbell noise. The role of the 'Now' argument is to pass the filename of the photo to the Gmail script. In other words it is the connector between our basic code, odroidbell.py, and the Odroid-IoTNotifier.py script. The OdroidC2.sh script is very simple:

```
<cd photos>
```

We change from present directory to the <photos> directory.

```
<echo "Taking the Photo">
<now=$1>
```

"Now" is the filename's timestamp.

```
<echo>?

<fswebcam -d /dev/video0 $now.
jpg>
```

This is the basic command for taking the photo. We use the fswebcam command. If the fswebcam application is not installed on your system you can install it with the following command:

```
$ sudo apt-get install fswebcam
```

The syntax of the command is obvious: it's taking a photo and passing the time stamp as a filename. With every button press, OdroidC2.sh is triggered by odroidbell.py. The -d switch sets the source to use, in our case /dev/video0.

```
<echo "Pic Taken">
<echo"">
<echo "Ringing Bell">
<echo "">
<cd ..>
```

Change back to the parent directory.

```
<mpv ringtone.mp3>
```

Finally we make that bell ring using a program called mpv, which is already included in Ubuntu 16.04 v2.0 from Hardkernel (http://bit.ly/2cBibbk). In other word we use the Mplayer to parse this file.

Gmail code setup

Most people already have a Gmail account. If you don't, it's very easy to create one, and most important of all, it's free. In actuality, in order for this Gmail script to work correctly, we need two email accounts: the sender's email and the recipient's email account and it's always like that. Of course, you can send an email to and from the same account, but it's more elegant to create a second email account for the purpose of this project to keep track of photos with the time stamp separately. I also recommend that the recipient's email account be the one used on your mobile phone so that the device will notify you whenever someone is at your door. Don't forget to allow "less secure apps" have access to your Gmail account (http://bit.ly/124TgWN).

Let's exam the python script called IoTOdroid.py.

```
<from email.mime.text import MIM-
EText>
<from email.mime.multipart import
MIMEMultipart>
```

We use those two modules because we need to send a clean email, with a sender, a receiver and a subject line.

```
<from email.mime.application im-
port MIMEApplication>
```

We also import the module responsible for the MIME attachment. MIME stands for Multipurpose Internet Mail

Extensions. It's a way of identifying files on the Internet according to their nature and format.

```
<import smtplib>
<from smtplib import SMTP>
```

This is the native and basic library in Python to send emails and therefore there is no need to install external libraries:smtplib. From this library, we import the SMTP function.

```
<import sys>
```

We import the System-specific parameters and functions module as we will need the <argv>script from this module (see below).

```
<recipients = ['<YourEmail>']>
```

Your email address, as you are the recipient of those photos.

```
<emaillist = [elem.strip().
split(',') for elem in recipi-
ents]>
```

We make an email list in which we strip and split the appropriate characters from each element in recipients list.

```
<msg = MIMEMultipart()>
```

We define the variable message (msg) as a Multipurpose Internet Mail Extensions by calling the MIMEMultipart function.

```
<msg['Subject'] = str(sys.
argv[1])>
```

The subject of our message.

```
<msg['From'] = '<From Email>'>
```

Your email address.

```
<msg['Reply-to'] = 'xyz@gmail.
com'>
```

The receiver's email address.

```
<msg.preamble = 'Multipart
massage.\n'>
```

The preamble attribute contains the leading extra-armor text for MIME documents, that's why we included it here.

```
<part = MIMEText("Hello! The
doorbell is ringing! A photo of
the person ringing the doorbell
has been attached.")>
```

The body of the message.

```
<msg.attach(part)>
<part =
MIMEApplication(open(str(sys.
argv[2]),"rb").read())>
<part.add_header('Content-
Disposition', 'attachment',
filename=str(sys.argv[2]))>
```

We attach the photo to the message

```
<server = smtplib.SMTP("smtp.
gmail.com:587")>
```

We specify the smtp server we want to use and the port that goes with it: Gmail's server with port 587. You can use port 465 too but, it's a good idea to check with Google for the correct number of ports, just in case something has changed.

```
<server.ehlo()>
```

We identify ODROID-C2 to the Google server. So, why do we use "ehlo" instead of "helo"? The reason is simple: we identify ODROID-C2 to an extended SMTP server (ESMTP) instead of a simple SMTP server, so the command "ehlo" distinguishes between the two.

```
<server.starttls()>
```

We set and we are entering to TLS mode. TLS stands for Transport Layer Security, so any SMTP command coming after this mode is encrypted.

```
<server.login('<From
Email>','<From password>')>
```

It's time to enter to our Gmail account, therefore we need the appropriate credentials.

```
<server.sendmail(msg['From'],
emaillist , msg.as_string())>
```

Using the previous command, we finally we send the email. Try to avoid spoofing and put your real email address here. The parameter, emaillist, is the one we defined before and the last parameter (msa.as_string()) the message as a string with the attachment, which is, in this case, the photo. The results are shown in Figure 3.

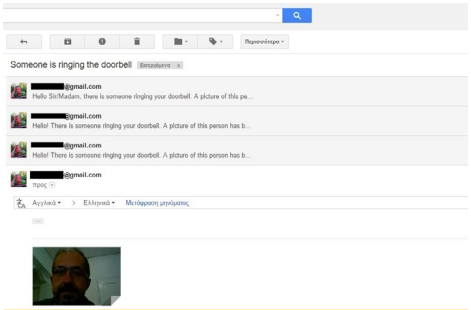


Figure 3 - Email alert that someone is at the door

Testing and running the code

From the terminal (CTRL-T), we run the odroidbell.py with sudo privileges:

```
$ sudo python odroidbell.py
```

The IoT device at this stage is set to standby mode, and the message "Program is running" appears on the screen. At the same time, the dots "." will appear on the screen, one by one, indicating to the user the normal usage of the device. When someone presses the button, the OdroidC2.sh script is executed with a

two-fold purpose: First, it takes a snapshot of the person ringing the bell with the characteristic doorbell ring. Second, it sends the email via the IoTOdroid.py to the owner's Gmail account with the timestamp of the photo attached. After that, it goes back to standby mode, and the IoT doorbell rings.

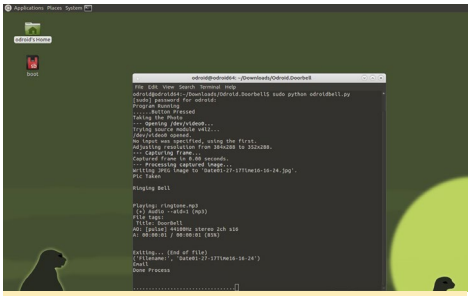


Figure 4 - Doorbell Python script running

Final notes

This project could certainly be enhanced in many ways. For example, a LED could be added on this IoT doorbell as an indicator of its proper usage. A more complex device, similar to this, could actually make use of a commercial doorbell with the aid of a relay board module. It's not also difficult to add the ability of a short video clip recording alongside the photo shots taken from the web camera, making the "smart" doorbell even smarter. Besides, as the old saying goes, "perfect is the enemy of good"!

IoT doorbell code

A copy of the code can be found in the following section, at the very end of this article is a link to the github page contain this code as well.

Odroidbell.py:

```
import wiringpi2 as odroid
import time
import os
import glob
import sys

odroid.wiringPiSetup()

Button = 3

odroid.pinMode(Button,0)
odroid.pullUpDnControl(Button,1)


#loop
print("Program Running")
while True:#loops forever till
keyboard interupt (ctr + C)
  if odroid.digitalRead(Button)
== False: #when button not
pressed:
    sys.stderr.write(".")
    time.sleep(1)
  else:
    print("Button Pressed")
    #    ------|    photo & Bell
|------ #
    #Get FileName
    now = time.strftime("Date%m-
%d-%yTime%H-%M-%S")
    #Make command to run
odroidC2.sh
    command = "bash odroidC2.sh "
+  str(now)

    # -- odroidC2.sh is an Shell
script that
    # -- is responsible for tak-
ing the photo and
    # -- making the Doorbell
Noise

    # --- We insert the "Now" ar-
gument so the python
    # --- script knows what the
file name of the
    # --- picture will be so it
can pass it on into the
    # --- email script (so it
knows what file to email

    #run command
    os.system(command)
    #diagnostics
    print("Filename:", now)


    # ----| Email     |---- #
    print("Email")#email
    emailcommand = 'sudo python
IoTOdroid.py "Someone is ringing
the doorbell"' + ' "photos/' +
now + '.jpg"'
    os.system(emailcommand) #run-
ning the Email script with:
    #-- the subject as "Someone
is ringing the doorebell" and the
filename
    #-- We made before at the
-Photo & Bell- section

    # -- End Diagnostic Info
    print("Done Process")
    #-space out for next "Press
of Button"
    print("")
    print("")
```

OdroidC2.sh:

```
#!/bin/sh
cd photos
echo "Taking the Photo"
now=$1 #Now is the filename time
stamp
#take pic
fswebcam -d /dev/video0 $now.jpg
echo "Pic Taken"
echo""
#ring Bell
echo "Ringing Bell"
echo ""
echo ""
cd ..
mpv ringtone.mp3
```

IoTOdroid.py:

```
from email.mime.text import MIM-
EText
from email.mime.application im-
port MIMEApplication
from email.mime.multipart import
MIMEMultipart
from smtplib import SMTP
import smtplib
import sys

recipients = ['abc@gmail.com']
emaillist = [elem.strip().
```

```
split(',') for elem in recipi-
ents]
msg = MIMEMultipart()
msg['Subject'] = str(sys.argv[1])
msg['From'] = 'xyz@gmail.com'
msg['Reply-to'] = 'abc@gmail.com'

msg.preamble = 'Multipart
massage.\n'

part = MIMEText("Hello! There is
someone ringing your doorbell.  A
picture of this person has been
atached.")
msg.attach(part)

part =
MIMEApplication(open(str(sys.
argv[2]),"rb").read())
part.add_header('Content-
Disposition', 'attachment',
filename=str(sys.argv[2]))
msg.attach(part)

server = smtplib.SMTP("smtp.
gmail.com:587")
server.ehlo()
server.starttls()
server.login('xyz@gmail.
com','yourpassword here')

server.sendmail(msg['From'],
emaillist , msg.as_string())
```

The project code is available for download at http://bit.ly/2jMAdMY using the following command:

```
$ git clone \
https://github.com/miltiadisme-
lissas/\
IoT.OdroidC2.Doorbell.git
```


LINEAGEOS-14.1 FOR ODROID-XU3/XU4

FORGET CYANOGEN, THE FUTURE IS HERE

by @voodik
edited by Bruno Doiche

Horror struck Cyanogenmod users at the end of 2016, when Cyanogen suddenly pulled the plug and left all users that relied on Cyanogenmod without support. However, a solution came in no time in the form of LineageOS. If you are a fellow ODROID-XU3/XU4 user, it is the perfect time to collaborate on the LineageOS release on the ODROID forums:

Features

- Android 7.1.1 Nougat LineageOS 14.1
- 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 Bluetooth support (BLE, A2dp Sink).
- USB GPS dongle support.
- USB tethering.
- Portable Wi-Fi hotspot.
- Android native USB DAC support
- USB UVC Webcam support
- HDMI-CEC support
- Selinux Enforce

Known issues

Only Bluetooth low energy v4.0(BLE) modules are supported at this moment. See Bluetooth Module 2

How to install.

For first time you need prepare your emmc/sd with spercial selfinstall images.

You can find it here

```
http://oph.mdrjr.net/voodik/5422/
ODROID-XU3/Android/CM-14.1-ATV/
Alpha-0.1_11.02.17/
```

Write the image to your eMMC/sd via Win32DiskImager and boot it up! You will need to wait patiently during the first booting process, since the update process might take up to 20 minutes.

Kernel Source

```
$ git clone https://github.com/\
voodik/android_kernel_hardkernel\
_odroidxu3 -b cm-14.0_5422
```

LINUX GAMING

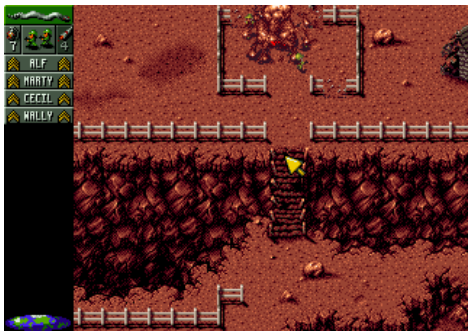
OPEN FODDER

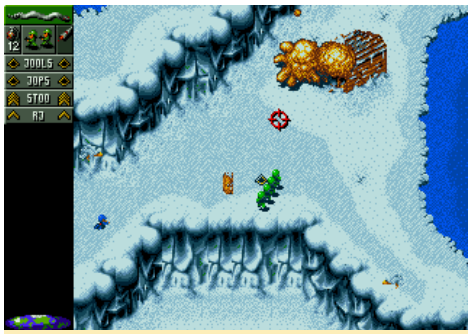
by Tobias Schaaf (@meveric)

This month, I'd like to talk about a game called Open Fodder, which is a remake of the classic Amiga game Cannon Fodder developed by Sensible Software way back in 1993. It uses the original game data from Cannon Fodder to bring a near-original experience to your ODROID.

Figure 1 - The Cannon Fodder logo, the game Open Fodder is based on

Many popular Linux games these days, such as Stratagus, and Freeciv, are in fact remakes of classic DOS games from the mid to late 1990s, and Open Fodder is no different. The Cannon Fodder remake is an action-strategy shoot 'em up game where you control a small group of soldiers that goes through dozens of levels to kill enemy soldiers, destroy tanks, blow up buildings, and defeat your enemies. The game was first released for Amiga way back in 1993, and was also ported by its developers to MS-DOS, the Atari Jaguar, SNES, 3DO, and Sega Genesis. The game had different settings to play in, such as jungle, snow, and desert.





Figures 2, 3 and 4 - A look at the Jungle, Desert, and Snow levels in Open Fodder

Open Fodder featured 23 missions that were each split up into several phases. In total, this meant 72 levels of leading your squad to victory. At the time of release, the game was highly praised by several Amiga gaming magazines, scoring as high as 95 percent and considered among the best games launched in 1993. Overall, it's still concerned one of the best games ever made for the Amiga platform during its 11-year lifetime.

On a more political note, the game also makes a rather stark commentary on war itself as you play and lead your soldiers to their inevitable deaths. Each mission brings up a fresh group of recruits that line up waiting to join the

Figures 5 and 6 - As you progress in the game, your long line of recruits gradually become crosses lining a cemetery

fight. The expendable soldiers (hence the name cannon fodder) gradually become crosses lined up across a military cemetery as you complete each mission, with more lining up just as you bury their predecessors! The developers certainly had a sense of humor though, as the very first soldiers in your squad (and hence those almost certain to die) are in fact bearing the six person team's own names.

On a different note, the game had some other interesting facts about it. For one, it has its own theme song, "War, never been so much fun", which plays during the intro of the game. One of the game's developers, Jon Hare, composed the music himself along with composer Richard Joseph! If you had the Amiga CD32, you may also remember a bonus track with a short movie featuring the developers shooting at each other with toy guns (http://bit.ly/2l67bFy). They certainly seemed to have a lot of fun making this game!

Cannon Fodder also had a few sequels, as well as some bonus missions released. Cannon Fodder 2 came out a year later, but was more of an expansion pack "data disc" than a sequel, as it mostly featured more missions, rather than new features. There was also the Amiga X-Mas special, which featured some different missions based on the Cannon Fodder game.

Playing Open Fodder

Open Fodder is a remake of the Cannon Fodder gaming engine, bringing the game to modern operating systems. Much like other game engine ports (like OpenTTD) you can use the original game data with this engine port to play the game on your modern device. This port also has a version ported to ARM-based devices, meaning we can run it on our ODROIDs too!

It's still under development, but is working quite nicely already. As usual it can be installed from my repository and since it only requires SDL2 and SDL2 Mixer as it's main dependency it should work on Debian and Ubuntu alike and can be found in my all/main package list for armhf and jessie/main for arm64.

It can be installed with this command on an ODROID device running Debian, assuming you have my repositories already configured, or if you're using one of my Debian images (http://bit.ly/13v98ly):

```
$ apt-get install \
  openfodder-odroid
```

I've slightly altered the game to always run in full-screen mode, but you can switch to Window mode simply by pressing F11 when you have the game running, if you want that.

Converting the Game Data

On its own, my version of Open Fodder comes with the Amiga X-Mas special, as well as the demo levels available for free. If you want to play the real game, then you'll need to import that data from a legitimate CD of Cannon Fodder.

If you're using an ODROID to play Open Fodder, you can find the game data folders in the $HOME/.openfodder/Data folder. For example, /home/odroid/.openfodder/Data/Dos_CD is one such game data folder. There are several game data folders you can use depending on which version your Cannon Fodder game is.

Dos_CD

Although the game supports many different game sources, currently only Dos_CD seems to be a "fully supported" Open Fodder version with the least bugs and issues. The other game sources may have glitches.

Here's a quick tip: Although it's called Dos_CD, it will take any DOS version of Cannon Fodder you can find. If you have the disk version of Cannon Fodder, just copy the CF_ENG.DAT into the Dos_CD folder. The original DOS CD version had a file called cf_cd.dat. Rename it to CF_ENG.DAT and copy it to the Dos_CD folder. If you happen to have the GoG version of the game, you also only need to copy CF_ENG.DAT over to the Dos_CD folder. This

Figure 7 - Sensible Software - The developers of Cannon Fodder

CAUSALITY

A TIME TRAVEL PARADOX PUZZLE GAME FOR YOUR DISCRETION

by Bruno Doiche

Set yourself amongside strange and alien landscapes, and help a group of stranded astronauts find a route to safety. Each level you face represents a hazardous moment where you need to complete it by taking your astronaut to an exit that matches their color. The timeframes are short but extremely fun to do it. Although you are participating in only a fraction of time of your character's lives, be warned: you will spend a bunch of time on your playing this game!

https://play.google.com/store/apps/details?id=com.lojugames.android.Causality

The isometric 3D environment is beautiful, and I often found myself failing the level because the graphics are so enthralling

is a quick and legal way to get the game if you're interested!

Make sure the name of the file is all capital letters, since the game is very picky about this. Try to match everything exactly to ensure things run smoothly.

Amiga_CD

Amiga CD32 had a version of this game as well. This is the version which included the video I linked above. On the Amiga CD32 CD, there is a folder called Fodder. The content of this folder needs to be copied over to the Amiga_CD folder. You can extract the second track of the CD and store it as Track2.flv in the same folder in order to have the video file.

I encountered some issues when I followed this, and later found again that the game searched for a lot of files in capital letters, so I used the following command to copy all files to have them in capital letters as well in the folder Amiga_CD:

```
$ for files in `ls`; do cp $files
`echo $files | tr '[:lower:]'
'[:upper:]'`; done
```

That way it was working for me, but the game had some glitches. For example, the helicopter animation on the start of each mission was missing, and the cursor when you save a file is somewhat messed up. Aside from that, I haven't seen any major issues in the Amiga CD32 version.

Amiga

The description from the Open Fodder developer says to, "Use the WHDLoad installer on an Amiga (or WinUAE) to extract the game files, and copy all the extracted files into the Data/Amiga folder". I haven't tested this, but if you have the unofficial Amiga CD32 "Cannon Fodder Collection", you can copy the content of the FodderNew in this folder, which will result in completely different levels. This feels like an extra hard version of the game, but it also fun to play. So either put the original Amiga files here, or the FodderNew files from the "Cannon Fodder Collection".

Dos2_CD

Dos2_CD is actually the folder for Cannon Fodder 2 DOS version, but it also works with the GoG version. Unfortunately, it has a few issues, the biggest being that the game doesn't have any sound or music at all. Aside from that, everything seems to be there, but I don't know how it reacts in later levels. The Data folder also comes with a WAV, Plus, AmigaFormat_XMAS and Custom folder. WAV is default sound effects for all games, (Cannon Fodder) Plus is a Demo from the Amiga Power magazine issue #31, and AmigaFormat_XMAS is the Amiga Format Christmas Special. There's also a Custom folder which allows you to play other games and maps as well, but they require the DOS (CD) version or else they are not even shown.

Final notes

Open Fodder is a nice project that allows you to play this awesome game on modern systems. I really like it, and I'm looking forward to see how it progresses. I hope the Amiga versions as well as Cannon Fodder 2 are soon fully supported. Maybe we can even see some graphical improvements in time. I wouldn't mind seeing updated graphics, since the game is rather old, and the graphics are not the best in 1080p.

REMOTEPI BOARD FOR THE ODROID-C2

by @inifity85

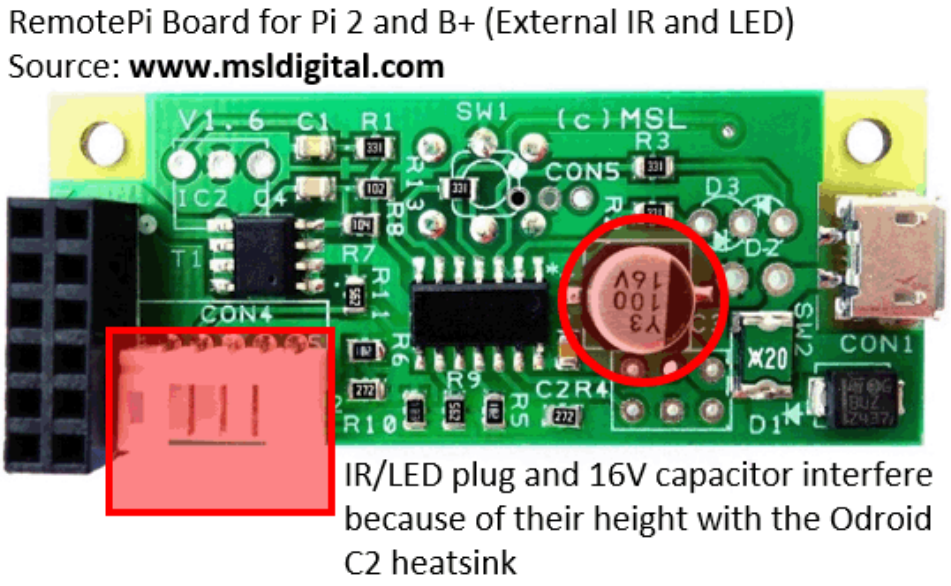


Figure 1 - RemotePi Board

The RemotePi Board (http://bit.ly/2l8JcWU), which turns any remote control into a power switch for your single board computer, can be made compatible with the ODROID-C2 if you follow this guide. If you use your device as a media center, this board provides an IR receiver and a power circuit as well as a power button. You can power on and off your ODROID completely via an arbitrary Infrared (IR) signal, and just push the power button for safe shutdown and reboot. The RemotePi Board is responsible for standby mode, which draw only some negligible power while the ODROID is completely off. Finally, the board powers your device through GPIO, so it requires a microUSB cable to be plugged into the RemotePi Board instead of using the normal power adapter.

Hardware configuration

Since the RemotePi Board was originally intended for the Raspberry Pi 2, you'll need to use additional wires instead of just putting it on top in order to avoid interfering with the heatsink, so you must rewire some pins. It's necessary to use wires of adequate diameter for the 5V and Ground pins, since the current could be 2A - 2.6A, depending on how many USB devices you connect to your device.

The RemotePi GPIO pins 8 and 10 need to be connected to different ones on the ODROID-C2, because there seems to be a conflict. The UART interface occupies these pins, and the ODROID's default state (1=high on Pin 8) is also not what the RemotePi Board expects (GPIO must go into default 0=low for cutting power after shutdown). But since you need to use cables anyway for connecting the RemotePi, this is not a big deal, since you can just repin the cables to other GPIO pins.

As we bypass the ODROID's power circuit by powering it through GPIO, the device's 2.5/2.6A over-current protection is also bypassed, but don't worry. The RemotePi also has over-current protection that should ideally match the same value, which is the case for the RemotePi Board for the Raspberry Pi 3. However, the RemotePi Board for the Raspberry Pi 2 is only 2A. This is fine, because the value is lower than the ODROID's 2.6A, so the RemotePi's polyfuse would trigger earlier if I attach too many non externally powered USB drives, which is not a good idea on an SBC anyway. However, because of this amperage difference, a RemotePi Board for the Raspberry Pi 3 would be the best choice for this project.

As a result of the rewiring, the RemotePi's firmware can no longer be configured the normal way, although I've never made use of this feature. If it becomes necessary to configure or update the firmware, you would need to rewire the RemotePi pins 8 and 10 to the ODROID's pins 8 and 10 during the firmware update, then back again after the update has been completed. Alternatively, you could just attach a Raspberry Pi on top to update the firmware. This sounds like a lot of trouble, but actually it's only extending the header connection and changing one line in the main script, and two lines in another optional script if you make use of it.

Rewiring the GPIO connection

Because the RemotePi is originally designed for a Raspberry Pi, it is unfor-

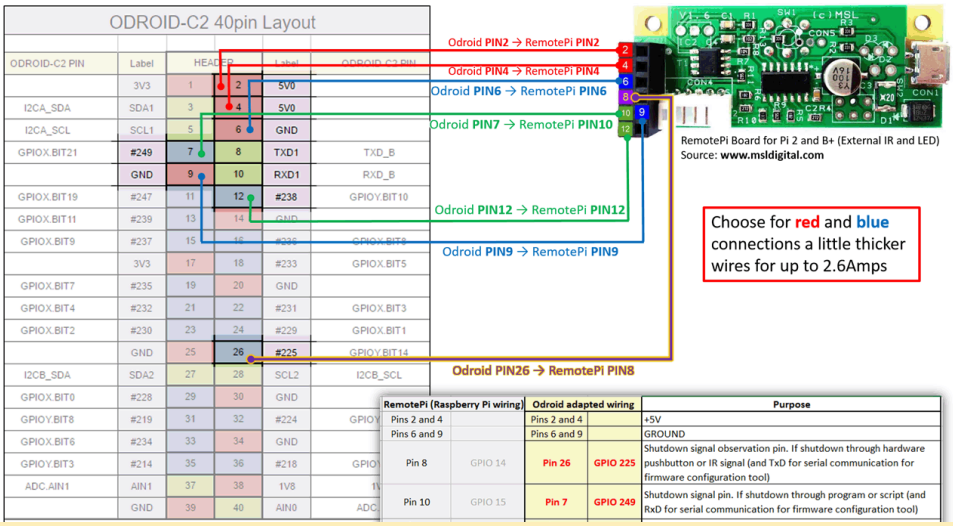
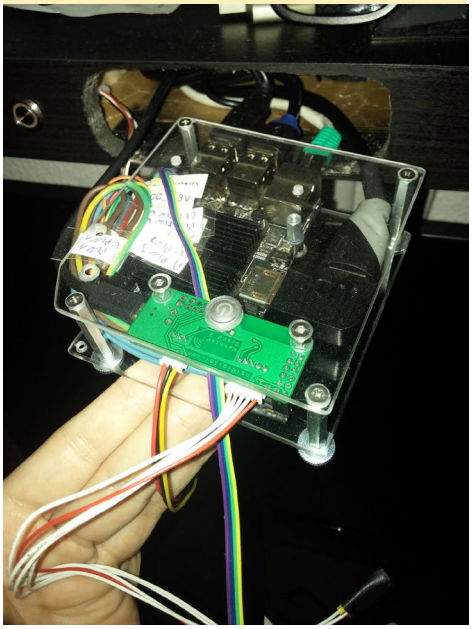


Figure 2 - GPIO rewiring

tunately not just plug and play. You'll need to rewire two pins besides using wires for all pins anyway, as shown in Figure 2. Rewire RemotePi PIN 8 to ODROID-C2 PIN 26, and RemotePi PIN 10 to ODROID-C2 PIN 7.

If you don't want to use the RemotePi Board IR receiver for remote controlling LibreELEC, then pin 12 can be left disconnected. The board will still power on and off via learned IR remote control commands, but controlling LibreELEC GUI will be taken over by the ODROID's built-in IR receiver. However, if you bought a RemotePi board with external LED and IR receiver, you may have chosen this for placing it somewhere in a custom case, so the built-in receiver might be hidden and useless. In that case, you can deactivate the ODROID IR in favour of the external GPIO-IR receiver on RemotePi Board, and connect pin 12. More information about switching to the GPIO-IR receiver in LibreELEC is available at http://bit.ly/2lpDl27.

Figure 3 - The RemotePi Board has been rewired and mounted to a custom case



Shutdown scripts for LibreELEC

The two scripts discussed below can be downloaded from the MSL Digital Solutions support page at http://bit.ly/2kMxyVG. That page also contains a guide for applying these scripts to other operating systems such as Volumio and RuneAudio.

The irswitch.sh script is for safe shutdown via IR-Remote and push-button. After pushing the button, the system will shut down safely, then RemotePi will wait for the GPIO225 to reach state 0 (low), which comes after a successful system safe shutdown. Finally, it will cut the power completely.

The shutdown.sh script is for safe shutdown via program interface or script. After navigating to the power-off button in GUI, the system will shut down safely, then RemotePi will wait for the GPIO225 to reach state 0 (low), which comes after a successful and safe system shutdown. Finally, it will cut the power completely.

irswitch.sh (just change the original MSL Digital script GPIOpin1=14 to GPIOpin1=225):

```
#!/bin/bash
# prevent restarting XBMC at
shutdown.  This is only used for
OpenElec before V5
LOCKDIR="/var/lock/"
LOCKFILE="xbmc.disabled"
# this is the GPIO pin receiving
the shut-down signal
# Raspberry Pi pin8: GPIOpin1=14;
Odroid-C2 pin26: GPIOpin1=225
GPIOpin1=225
# functions
add_omit_pids() {
omit_pids="$omit_pids -o $1"
}
safe_shutdown () {
# for OpenElec before V5
touch "$LOCKDIR/$LOCKFILE"
# for OpenElec V5 and later
systemctl stop kodi
add_omit_pids $(pidof connmand)
add_omit_pids $(pidof dbus-dae-
mon)
killall5 -15 $omit_pids
for seq in `seq 1 10` ; do
usleep 500000
clear > /dev/tty1
killall5 -18 $omit_pids || break
done
sync
umount -a >/dev/null 2>&1
poweroff -f
}

echo "$GPIOpin1" > /sys/class/
gpio/export
echo "in" > /sys/class/gpio/
gpio$GPIOpin1/direction
while true; do
  sleep 1
  power=$(cat /sys/class/gpio/
gpio$GPIOpin1/value)
  if [ $power != 0 ]; then
    echo "out" > /sys/class/gpio/
gpio$GPIOpin1/direction
```

```
    echo "1" > /sys/class/gpio/
gpio$GPIOpin1/value
    sleep 3
    safe_shutdown
  fi
done
```

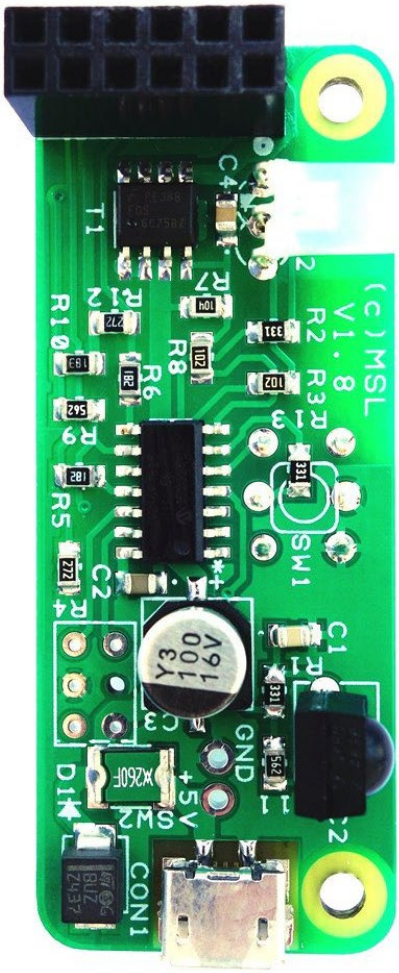
shutdown.sh (just change GPIOpin=15 to GPIOpin=249 and GPIOpin1=14 to GPIOpin1=225):

```
#!/bin/bash
if [ "$1" != "reboot" ]; then
# Raspberry Pi pin10: GPIOpin=15;
Odroid-C2 pin7: GPIOpin=249
  GPIOpin=249
# Raspberry Pi pin8: GPIOpin1=14;
Odroid-C2 pin26: GPIOpin1=225
  GPIOpin1=225
  echo "$GPIOpin" > /sys/class/
gpio/export
  # execute shutdown sequence on
pin
  echo "out" > /sys/class/gpio/
gpio$GPIOpin/direction
  echo "1" > /sys/class/gpio/
gpio$GPIOpin/value
  usleep 125000
  echo "0" > /sys/class/gpio/
gpio$GPIOpin/value
  usleep 200000
  echo "1" > /sys/class/gpio/
gpio$GPIOpin/value
  usleep 400000
  echo "0" > /sys/class/gpio/
gpio$GPIOpin/value
  # set GPIO 14 high to feedback
shutdown to RemotePi Board
  # because the irswitch.sh has
already been terminated
  echo "$GPIOpin1" > /sys/class/
gpio/export
  echo "out" > /sys/class/gpio/
gpio$GPIOpin1/direction
  echo "1" > /sys/class/gpio/
gpio$GPIOpin1/value
  usleep 4000000
fi
```

The shutdown.sh script is useful if you sometimes use Yatse or Kore android remote apps or hotkeys to shutdown your Media Center, for example. Those shutdown commands equate to internal events, similar to navigating to the Kodi shutdown menu. Without using this second script, the system would safely shut down, but the RemotePi Board would not receive any indication to monitor GPIO225, thus it would not cut off power after a successful shutdown.

Using the RemotePi Board IR receiver

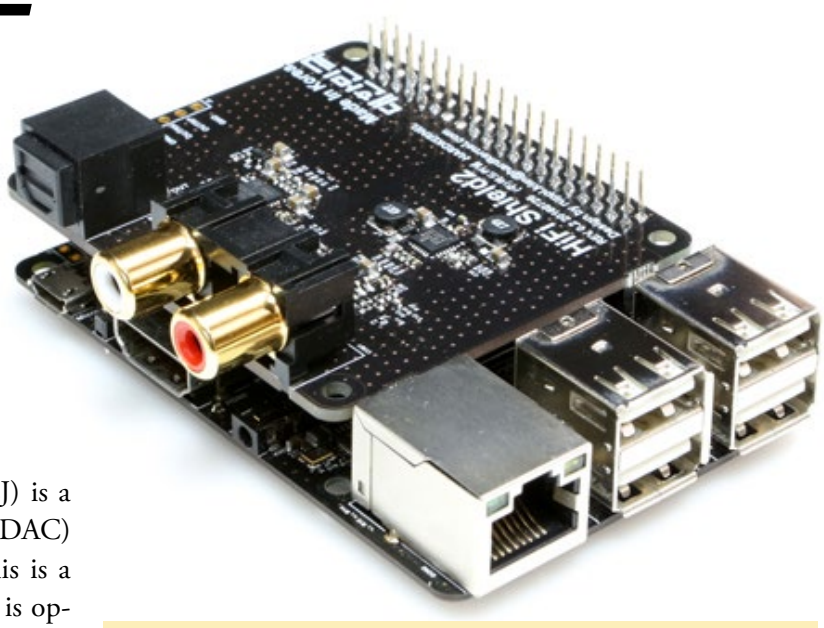
If you want to use the RemotePi's IR receiver instead of the onboard ODROID IR receiver, you'll have to deactivate the built-in IR and activate the GPIO IR receiver. To do this in Ubuntu, you can refer to Hardkernel's Wiki article at http://bit.ly/2l8KrWg. For LibreELEC, read my mini-guide at http://bit.ly/2lLKj2A. For questions, comments or suggestions, please visit the original thread at http://bit.ly/2mgFGKk.



HIFI SHIELD 2

THE BEST AUDIO YOU CAN ACHIEVE ON AN ODROID

edited by Rob Roy (@robroy)

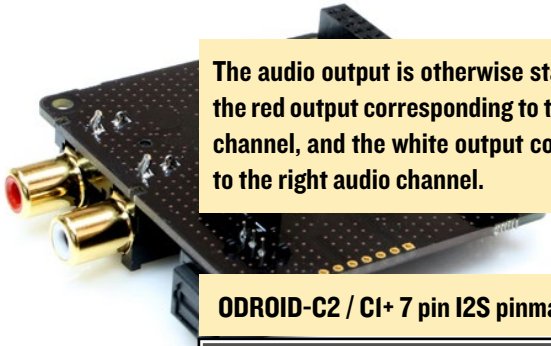


ODROID-C2 and HIFI Shield 2: Audio PB +J

The HiFi Shield 2 (USD$39 http://bit.ly/2lHSlZJ) is a high-resolution Digital to Analog Converter (DAC) for the ODROID-C2 and ODROID-C1+. This is a special sound card for the ODROID-C2 and C1+ that is optimized for the best fidelity audio playback quality. It delivers a nicely balanced sound: solid, deep, wide and nicely layered. We've analyzed the the audio quality of the HiFi Shield 2 DAC output with the famous industry standard equipment called Audio Precision. The Audio Precision is a high performance audio analyzer optimized for the digital audio product.

Using Texas Instrument's high-end PCM5242 DAC chip, known as Burr-Brown, the HiFi Shield 2 supports 16, 24, 32 bit audio formats with minimal THD+N ratio (0.002%) and ideal dynamics (114dB+), plus amazing sampling rates of 384kHz. One dedicated S/PDIF interface supports up to 192kHz/24bit resolution via an optical (Toslink) output.

Using the I2S expansion port on the C2 / C1+, it doesn't need to occupy a USB port, and allows the user to select their audio playback system of choice, such as Volumio and Debian (DietPi) for HiFi audio playback.

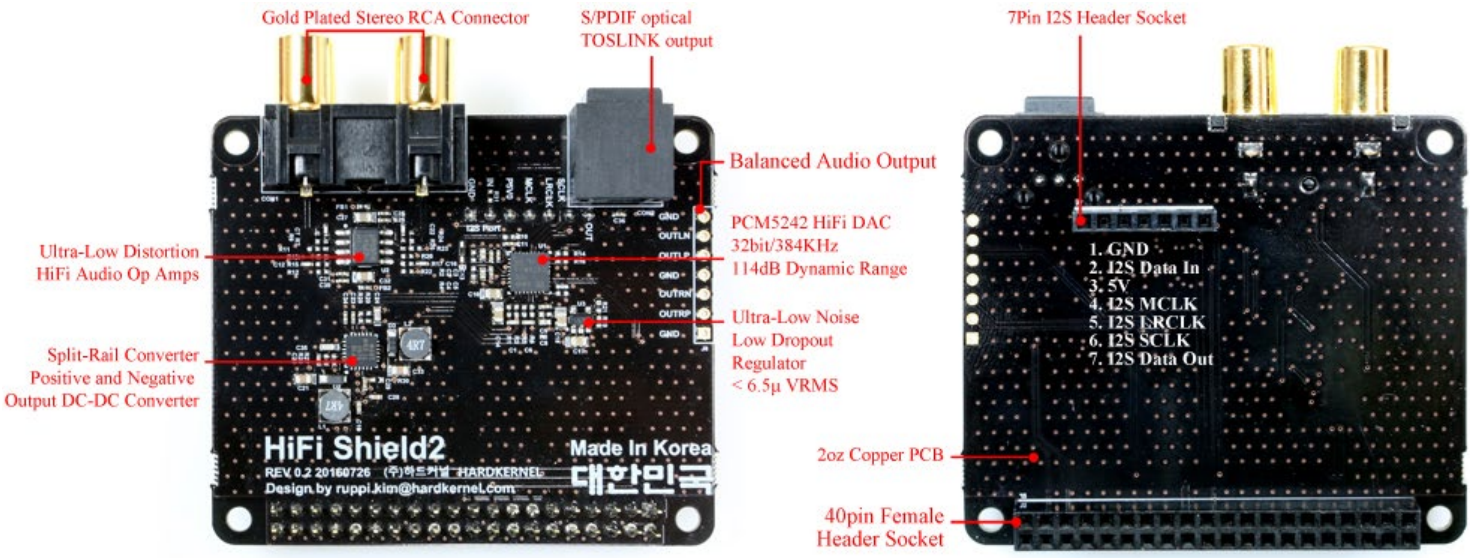


The audio output is otherwise standard, with the red output corresponding to the left audio channel, and the white output corresponding to the right audio channel.

ODROID-C2 / C1+ 7 pin I2S pinmap

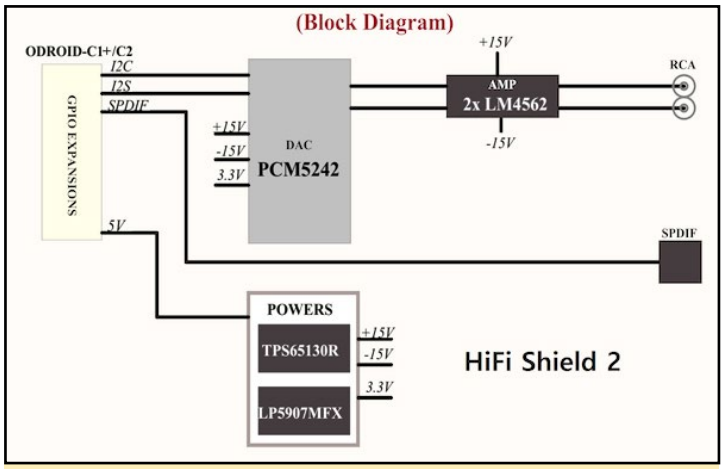
ODROID-C2/C1+ 7pin Layout	
HEADER	Label
1	GND
2	I2S Data In (for ADC)
3	5V
4	I2S MCLK (Master Clock)
5	I2S LRCLK (LR Clock)
6	I2S SCLK (Bit Clock)
7	I2S Data Out (for DAC)

Annotated board closeup



Features

- Volume control via I2C interface is great feature to keep great audio quality with various audio output volume
- The output ports include gold-plated stereo RCA terminals
- An ultra-low noise dropout regulator is populated for the power supply, significantly reducing power supply noise and greatly increasing the signal to noise ratio
- The I2S interface allows for direct decoding of the digital input to analog output using master clock synchronization
- The PCB surface is comprised of gold-plating on top of 2 oz of copper, ensuring signal continuity and reducing signal reflection and refraction
- Balanced (differential signal) audio output soldering pads are available
- The dedicated S/PDIF interface is new, which supports up to 192kHz/24bit resolution via a new Optical (Toslink) output
- This HiFi shield is not compatible with Android, and Hardkernel has no plan to support the Android OS to enable the I2S driver for the Android Kernel and HAL



HiFi Shield2 Block Diagram

Details

The Ubuntu/Linux setup guide for ODROID-C2 is available at http://bit.ly/2brrGdG, and for the ODROID-C1+ at http://bit.ly/2lI7AC7. The official Volumio 2 operating system can be downloaded from http://bit.ly/2kOJNAV, and the Debian-based DietPi image is at http://bit.ly/2ls45yM. You can view the schematics at http://bit.ly/2mnukV4.

UPDATED XU4 MANUAL

REVISED FOR UBUNTU 16.04 AND NEWER PERIPHERALS

edited by Rob Roy (@robroy)

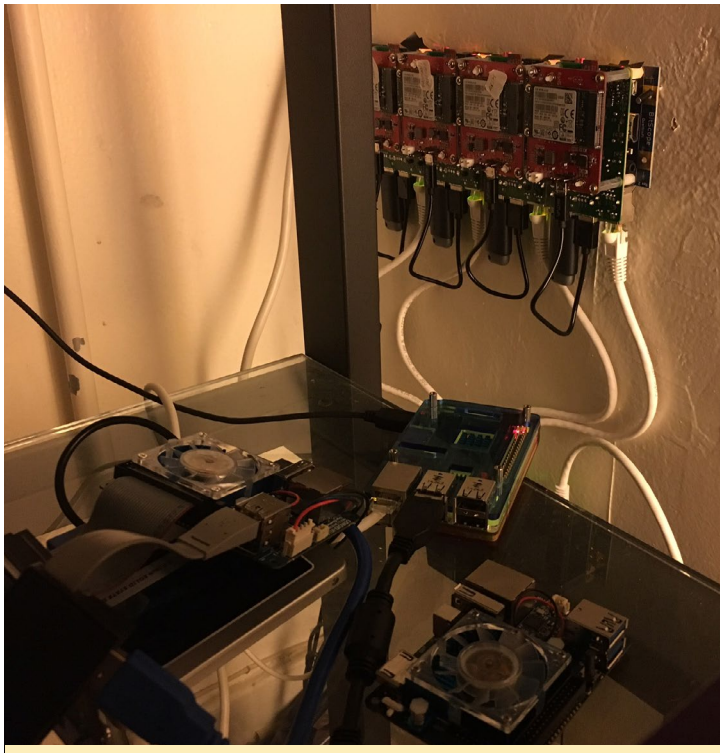
The ODROID-XU4 User Manual, available for download at http://bit.ly/1U9Q8yg, has been revised recently to reflect some of the newer peripherals, such as the Expansion Board, SmartPower2, and oCam. Since Hardkernel also now offers Ubuntu 16.04, all of the code examples were updated to be compatible with the new operating system. If you have feedback, questions, or suggestions, visit the ODROID forum thread at http://bit.ly/1RykBrT.

The ODROID-XU4 manual gives detailed explanations of the different operating systems, software, and peripherals available for the ODROID-XU4

HOME DATA CENTER

CODE DEPLOYMENT WITH ARCHLINUX

by John Skilbeck



John's home data center is a work of art

DevOps is hard. Large software projects, such as Mesos and Kubernetes, devops teams as found in most tech organizations, and companies such as CoreOS exist to support developers in getting their apps out the door and running. However, what's a good solution for a lone developer, or a small home network? How can we use code as infrastructure?

I use a Macintosh OSX laptop for development, but for long-lived applications or apps to run at night, I need a remote always-on environment, since my laptop will be offline or on the train with me while commuting. It makes sense to not use the development machine as an environment for deployment.

The ODROID-XU4 is an ideal computer for a remote deployment environment, because it's inexpensive, flexible, has great specifications, and can run Linux. The goal of this article is to explain how to keep deployment code in your project repository, as well as automate deployments and execution.

Arch Linux

Arch Linux is a free and open-source Linux distribution that was first released in 2002. It focuses on elegance, code correctness, minimalism and simplicity, and expects the user to make some effort to understand the system's operation. Arch Linux notably uses a rolling release model, so all that is needed to obtain the latest system software is a regular system update.

Arch Linux can be difficult to pick up, as it uses different tools than a Debian distribution. The package manager is called via "pacman" rather than "apt-get", and there is a popular add-on package manager called "yaourt". Many common tools or services aren't installed by default.

Arch Linux is mainly for x86 processors, but a project called Arch Linux ARM (ALARM) has an ARM distribution of Arch Linux for ARMv7 and ARMv8 AArch64 architectures. Hardkernel, the manufacturer of ODROIDs, is actually a sponsor of the Arch Linux ARM project.

Network Setup

You'll want to give your device a reserved DHCP LAN IP address, and ideally a hostname that gets propagated over your network through your router's DNS server. That way, on our development/local environment, we can use a hostname to always resolve to the remote/deployment environment.

For example, on my network I reserve 192.168.2.49 to my ODROID's MAC address. I also setup a DNS entry that maps that IP address to "odroid". Using a custom router firmware like Tomato USB or DD-WRT makes this extremely easy, since those firmwares turn your router into a small Linux computer with a slick GUI webapp, but implementing that is outside the scope of this article. If going across subnets, make sure to port forward an external port that maps to the SSH port of the odroid device, since Git runs over SSH.

Project Setup

Logically, you'll want to standardize your deploy workflow. That will make working across projects extremely easy, and remove a lot of the mental context switching you use when working across projects. We'll create a folder for housing all of our files related to deployments. Place any executable files in "deploy/bin", and any cron files in "deploy/tasks" (more on that later).

Navigate to the project directory in a Terminal window, then type the following commands:

```
$ mkdir -p deploy/bin
$ mkdir -p deploy/tasks
$ cd deploy/bin && touch run-job && \
  chmod u+x run-job && cd -
$ cd deploy/tasks && touch crontab
```

You could also standardize where you place your source code. That way, it becomes very easy for another party to see how your project is organized, and to know what is source code and what is not.

```
$ mkdir src
$ cd src && (place source code here, ie python: core.
py, clojure: core.clj, nodejs: app.js)
```

Abstracting the entry point

Starting an app can be confusing with all the various commands to run in various languages. For example, you might use Java "java -jar [my-jar].jar", or python "python my-app.py", and your app also might require various arguments. This should all be simplified into a single file and abstracted into a file "deploy/bin/run-job":

```
#!/bin/sh
set -e
CMD="src/duck"
exec $CMD $@
```

Creating the cron file

Arch Linux doesn't come with a cron daemon or client by default. Install it with "sudo pacman -Syu cronie". Using cron, you can run commands at specified time intervals using special cron syntax. This is normally stored in a user's crontab file, which you access with "crontab -e". However, this is too manual, and we want to use code as infrastructure. Cron also has some helpful subdirectories in "/etc/cron.*", like "/etc/cron.daily" and "/etc/cron.hourly", and if we place files here, they will be run at those intervals specified.

Review the file from "deploy/tasks/crontab" that we'll place in "/etc/cron.d", which is done automatically with our "post-receive" script:

```
### variables ##
SHELL=/bin/bash
PATH=:/bin:/usr/bin:/usr/local/bin:/usr/sbin:/usr/lo-
cal/sbin
MAILTO=[your-email-address]@gmail.com
cmd="deploy/bin/run-job"

app_dir="/home/skilbjo/deploy/app/duckdns"

## jobs ##
5 * * * * skilbjo cd "$app_dir" ; $cmd >/dev/null
```

Here is an overview of a simple project structure. The only project executable is a single shell script in "src/":

```
$ tree
.
├── README.md
├── deploy
│   ├── bin
│   │   ├── post-receive
│   │   └── run-job
│   └── tasks
│       └── crontab
└── src
    └── duck

4 directories, 5 files
```

Git

First we'll want to add a remote url in our project on our local environment:

```
$ git remote add odroid ssh://odroid/~/deploy/git/
duckdns.git
```

Note that depending on your network topology, this url might need to be adjusted. If you can't assign hostnames, then the git url would look like this, where 192.168.2.49 is your device's LAN IP address:

```
$ ssh://192.168.2.49/~/deploy/git/duckdns.git
```

If you have a different user in your ODROID environment than your development environment, then the url would look like this, where "skilbjo" is your username:

```
$ ssh://skilbjo@odroid/~/deploy/git/duckdns.git
```

If your remote server is on a different subnet and you are portforwarding, your url would

look like this, where "2222" is your external port:

```
$ ssh://192.168.1.2:2222/~/deploy/git/duckdns.git
```

In the home directory of your remote environment, create a folder called "~/deploy" with

two subfolders: "~/deploy/app" and "~/deploy/git". The

"~/deploy/git" subdirectories will be the endpoints of our pushes, and with a hook, they will then execute some deployment commands in the "~/deploy/app" subdirectories.

First, navigate to the home directory of the remote environment, then type the following commands:

```
$ mkdir -p ~/deploy/app
$ mkdir -p ~/deploy/git
$ mkdir -p ~/deploy/git/duckdns.git
$ mkdir -p ~/deploy/app/duckdns
```

Now, in "~/deploy/git/duckdns.git/hooks", create an executable file called "post-receive", which will be triggered with every push to the endpoint.

```
$ cd ~/deploy/app/git/duckdns.git/hooks
$ touch post-receive && chmod u+x post-receive
$ vim post-receive
```

Fill the executable with the following in the remote environment's "~/deploy/git/duckdns.git/hooks" directory:

```
#!/usr/bin/env bash
set -eou pipefail

user=$(whoami)
dir="/home/${user}/deploy/app"
app=$(basename $(pwd) | sed -e 's/.git//')
deploy_dir="$dir/$app"
cron_dir="/etc/cron.d"

GIT_WORK_TREE="$deploy_dir" git checkout -f

cd "$deploy_dir"

## build steps here ##
case "$user" in
  (skilbjo) sudo cp deploy/tasks/crontab "$cron_
dir/$app" ;;
esac

## you can also do project-specific build steps in
this section, like install
## dependencies, ## (ie npm install), compile source
code (ie lein uberjar),
## as well as if a long-lived app, run commands as
well (ie java -jar my_jar.jar)

echo "all done"

exit 0
```

Deployment

Now we're ready to deploy because our local environment is setup to hit the deployment server's endpoint, our remote environment is setup to receive the notification and check out the source code, run any build steps, and place a job in the system's cron directory for launching. Deploy everything with the following command in the local environment's project directory:

```
$ git push odroid
```

Additionally, to see how this has been implemented in a sample project, visit http://bit.ly/2lthYKW.

Next steps

Some features that can be added to the flow above are multiple environments, either with

multiple ODROIDs, or with a single ODROID treating it as a utility server. This can be done

with subdirectories under "~/deploy" such as "~/deploy/staging/app/my_app" or "~/deploy/production/app/my_app".

Additionally, you could add a continuous integration service like CircleCI that would run a test suite from every push to GitHub, and if successful, build a Docker image. You could then have have a file in the remote environment that would checkout an image from a Docker repository and run that image at a specified interval. This is what many of the distributed DevOps softwares do, such as Mesos and Kubernetes, but in a much more feature rich environment than bare-bones BASH and Linux.

References

Arch Linux Wikipedia article http://bit.ly/2l71ADK
Arch Linux ARM home page https://archlinuxarm.org
Article that I read three years ago that inspired the idea of writing my own article http://bit.ly/2m5JAct

THE ODROID ARCADE BOX

HAVE THE PERFECT EXPERIENCE WITH YOUR FAVORITE ARCADE GAMES

by Brian Kim, Charles Park and John Lee

ODROIDs have better performance than the competitor boards, especially in video rendering, which means that ODROID boards are very suitable for playing games, which many ODROID users do. There are already several game platform operating systems available, such as Lakka (http://bit.ly/1NO8BBC) and ODROID GameStation Turbo (http://bit.ly/1ASFO5O). In order to enjoy our gaming sessions more, we made our own arcade box with simple GPIO buttons and joysticks, and called it the ODROID Arcade Box. We choose an ODROID-XU4 for this project because it has the best GPU performance of all the current ODROID devices. This article describes how to recreate the ODROID Arcade Box for yourself.

Our first simple prototype

Requirements

(Figure 3 –)

We decided to make the ODROID Arcade Box using MDF (Medium-Density Fibreboard). The XU4 Shifter Shield is also useful in order to utilize the expansion pins of the ODROID-XU4. Joysticks, buttons and cables were

the input components, and an SMPS (Switched-Mode Power Supply) was used for the power supply. The detailed tools and parts list are listed here:

The ODROID Arcade Box needs a total of 27 inputs (19 inputs for buttons and 8 inputs for joysticks). The ODROID-XU4's digital 24 GPIO inputs are not sufficient for all 27 inputs, so we created two additional ADC ports for the three additional buttons. The ADC input values are based on input voltage, and the digital and analog input values are processed in the GPIO key daemon, which is described below.

Design and assembly

The panels of the ODROID Arcade Box must be designed and manufactured so that the buttons and joysticks are well placed. We chose 12T MDF considered for price and durability. Your design can be done with any familiar CAD tool such as Google Sketch or SolidWorks. Although there are many layout templates for joypad panels available, we chose a standard Japanese arcade layout.

The first step of assembly is to attach the sheet to the MDF panel. This step was easy, but took longer than the other steps. After that, we inserted the joysticks, power socket, switch and buttons on the top MDF panel. The HDMI, Ethernet and USB extenders were inserted on the back of the MDF panel. The next step was to assemble each MDF panel by using a drill to make holes in it, then using screws to hold it together.

The last step of assembling the ODROID Arcade Box was wiring the ODROID-XU4 expansion pins to the input components. In this project, we designed the external GPIO inputs, as shown in page 26. The Select and Temp buttons are connected to ADC expansion ports, as shown on page 26 too.

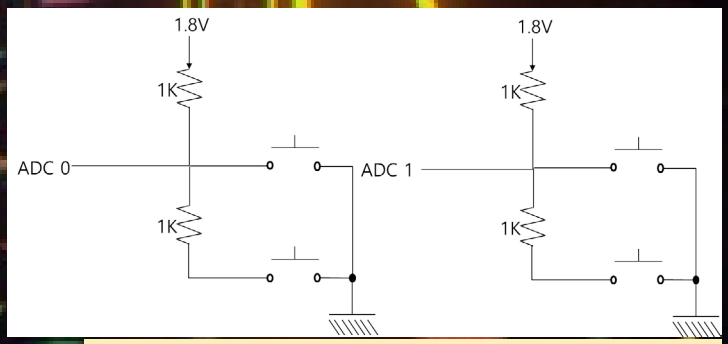
Software Setup

We developed a new GPIO key daemon called gpio_keyd (http://bit.ly/2ljOZKg). The gpio_keyd daemon is able to map GPIO inputs and key events using uinput and wiringPi, which is a pin-based GPIO access library. It's designed to be familiar to people who have used the Arduino wiring system. Although the upstream wiringPi library supports only Raspberry Pi, Hardkernel offers a wiringPi fork for ODROIDs in its GitHub repository (http://bit.ly/1Eq3UpF). The module uinput is a Linux kernel module that handles the input subsystem from user land. It can be used to create and handle input devices from an application.

We choose ODROID GameStation Turbo (http://bit.

Tools , parts and what they are just below:

12T MDF Panel	SMPS
2EA 600x220	HDMI, USB, Ethernet extenders
2EA 600x75	Power socket & Switch
2EA 220x75	2EA Hinges
Drill	Door catcher
Crimper	4EA foot rubber
Stripper	Screws
Measuring tape	19EA Buttons
Utility knife	2EA Joystick
Long Nose Plier	Wires
ODROID-XU4	Terminals
XU4 Shifter shield	



Expansion ports schematic

ly/1ASFO5O) as the software platform for our ODROID Arcade Box, which has uinput built in. You should make sure the uinput device file exists in your chosen operating system, because some of them do not have uinput devices.

```
$ ls /dev/uinput
```

If your operating system does not have a /dev/uinput device file, then it will be necessary to rebuild and install a new kernel with the INPUT_UINPUT option configuration option set. The Wiki page at http://bit.ly/1YIToBI describes how to build and install the kernel image from source code.

```
$ make menuconfig
Device Drivers
-> Input device support
    -> Generic input layer
      -> Miscellaneous device
        -> User level driver support <*>
```

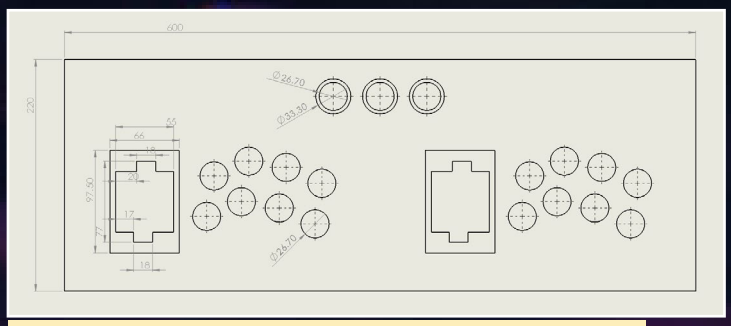
Note that wiringPi must be installed before installing gpio_keyd. On the ODROID GameStation image, the sudo commands must be run as root, because the "odroid" account is not designated as a sudo user.

```
$ git clone https://github.com/hardkernel/wiringPi.
git
$ cd wiringPi
$ sudo ./build
```

Download the gpio_keyd source code, which is available from our GitHub repository. The gpio_keyd build and installation methods are very simple:

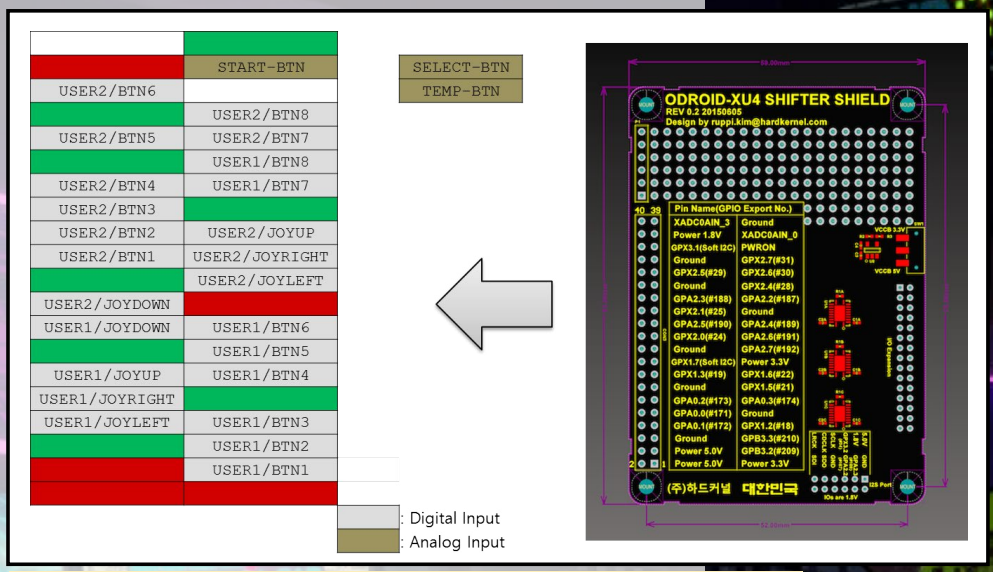
```
$ git clone https://github.com/bkrepo/gpio_keyd.git
$ cd gpio_keyd
$ make
$ sudo make install
```

The gpio_keyd script refers to /etc/gpio_keyd.conf as the default for GPIO and key mapping information. The configuration file was modified for 27 inputs of ODROID Arcade Box. Some keys are already used in the game emulator, so we had to change the emulator key settings in order to avoid key collisions between the emulator and GPIO input keys. Note that <GPIO pin> field in the configuration file refers to the wiringPi number, not the GPIO and pin number (http://bit.ly/2lbzPIB).



Joypad Layout Blueprint

Assembled ODROID Arcade Box Outline



External GPIO mappings for the Buttons and Joysticks

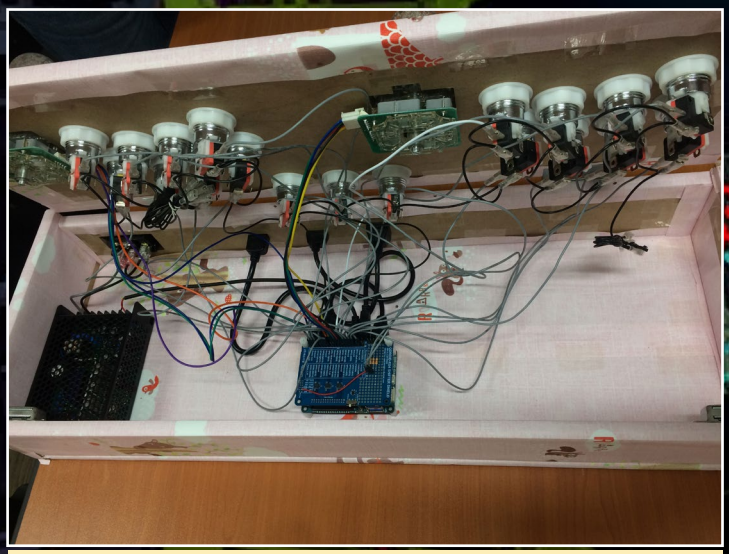
Configuration file sample for 27 inputs: /etc/gpio_keyd.conf

```
# Digital input
# <Key code>  <GPIO type>     <GPIO pin>      <Active
value>
# User 1
KEY_LEFT      digital         15              0
KEY_RIGHT     digital         1               0
KEY_UP        digital         4               0
KEY_DOWN      digital         16              0
KEY_A         digital         2               0
KEY_S         digital         3               0
KEY_D         digital         30              0
KEY_F         digital         21              0
KEY_Z         digital         8               0
KEY_X         digital         9               0
KEY_C         digital         7               0
KEY_V         digital         0               0
# User 2
KEY_BACKSLASH digital         12              0
KEY_SLASH             digital         13              0
KEY_SEMICOLON digital         14              0
KEY_LEFTBRACE digital         5               0
KEY_Y         digital         26              0
KEY_U         digital         27              0
KEY_I         digital         22              0
KEY_O         digital         23              0
KEY_H         digital         6               0
KEY_J         digital         10              0
KEY_K         digital         11              0
KEY_L         digital         31              0

# Analog input
# <Key code>  <GPIO type>     <ADC port>      <ADC ac-
tive value>
KEY_B         analog          0               0
KEY_N         analog          0               2045
KEY_M         analog          1               2045

To run gpio_keyd daemon at every startup is conve-
nient for ODROID Arcade Box.

/etc/init.d/gpio_keyd
#! /bin/sh
```



ODROID Arcade Box Wiring

```
### BEGIN INIT INFO
# Provides:          gpio_keyd
# Required-Start:    $all
# Required-Stop:
# Default-Start:     2 3 4 5
# Default-Stop:
# Short-Description: Run /usr/bin/gpio_keyd if it ex-
ist
### END INIT INFO


PATH=/sbin:/usr/sbin:/bin:/usr/bin

.  /lib/init/vars.sh
.  /lib/lsb/init-functions

do_start() {
      if [ -x /usr/bin/gpio_keyd ]; then
              /usr/bin/gpio_keyd -d
              ES=$?
              [ "$VERBOSE" != no ] && log_end_msg $ES
              return $ES
      fi
}

case "$1" in
    start)
      do_start
        ;;
    restart|reload|force-reload)
        echo "Error: argument '$1' not supported" >&2
        exit 3
        ;;
    stop)
      killall gpio_keyd
        exit 0
        ;;
    *)
        echo "Usage: $0 start|stop" >&2
        exit 3
        ;;
Esac

$ sudo chmod +x /etc/init.d/gpio_keyd
$ sudo update-rc.d
gpio_keyd defaults
$ sudo reboot
```

In the above commands, the gpio_keyd script runs as a daemon using the "-d" option. Usage of gpio_keyd can be checked with the "–h" option. Double-check the keys used by the game or the emulator, then set the gpio_keyd settings correctly. Then, you are ready to play and enjoy your games with your new ODROID Arcade Box.

The King of Fighters 98, John vs. Brian

ANDROID DEVELOPMENT

ANALYZING APPLICATION NETWORK USAGE

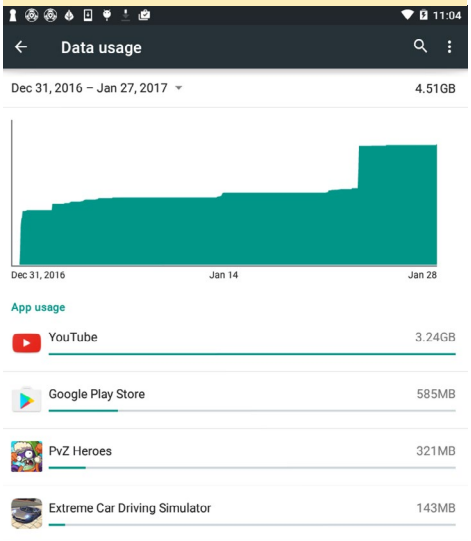
by Nanik Tolaram

As a developer, we want our apps to be efficient and often we would like to know how much network bandwidth our app is using. This is useful for a number of reasons:

- To monitor and make sure that our app is really our app and not app that has been hacked and published under different name in the play store
- To make sure we are not taking unnecessary bandwidth of user's data plan
- To ensure the device that we are using is not accessing the internet without our knowledge

In this article, we will look into the different ways you can look into getting the network usage data.

Figure 1 - Data usage inside Settings



Settings

The normal way we analyze network traffic data is by using the Data Usage via the Settings apps as shown in Figure 1.

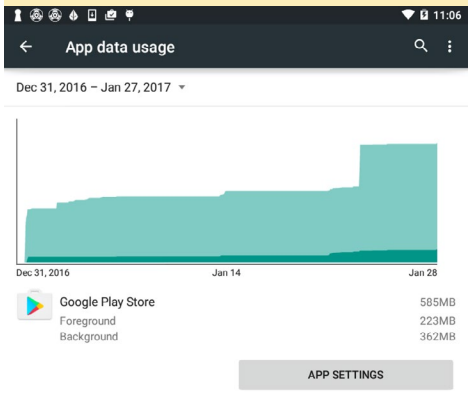
The data usage shows the total incoming and outgoing amount of data traffic that has been used by an application. If you select the application, you will see a screen like Figure 2 that shows more detail information about foreground and background usage of the application.

Information shown inside the Settings app are stored inside /data/system/netstats, which requires root access. By removing everything inside this folder, one can reset the network stats shown in the Settings app.

Network Stats

The Settings app can give us a high level overview of the data network stats, which is a good start, but sometimes we require more detailed analysis, which

Figure 2 - App data usage breakdown



can be done via the dumpsys command. Android provides a powerful tool call dumpsys that allow us to get a snapshot or dump of the system, which can include information about the network, memory and other components. Read through the Android documentation at http://bit.ly/2kK9dep to get more system information. For network related information, we would be interested in the command:

```
$ dumpsys netstats detail
```

Figure 3 shows a snapshot of what can be seen from a Nexus 7 tablet running Lollipop 5.1.1.

There are a few things that are important to understand about the stats:

- UID stats shows the breakdown for apps into background and foreground.
- uid shows the userid of the application which can be used to correlate with the package information (which we will see in the dumpsys package detail section) to know which application these network stats are for.
- tag is useful if you want to see how much data each connection in your application is using. An example of this can be seen in Figure 4, which shows uid 10007 has the following different tags:

Figure 3 - Dumpsys netstats detail

0x30000401, 0x30003201, 0x30002804, 0x10000401 and 0x30000407. These different tags help the developer know how much data each part of the application is using. This particular network stats output is linked to the Google Framework package which is part of the Google Play Store.

We will see in the next section how to write an application by tagging the thread that makes the network connection, to determine how much data is used.

Package Stats

In the previous section, I discussed the uid that is found when running the dumpsys netstats command. Now we need to know which uid corresponds to which application in the device and this is done by using the following command:

```
$ dumpsys package detail
```

This command will give a long list of information about the packages that are installed in your device. For the network stats, we are interested in the section shown in Figure 5.

The highlighted section shows the userId that relates to the uid shown in network stats. By relating this number, we know the package name of the application that the network stats we are reading.

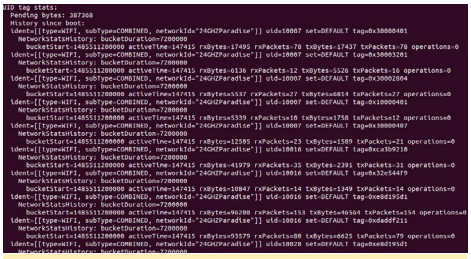


Figure 4 - UID tag stats

Figure 5 - dumpsys package detail

Sample app

The sample app for this article can be downloaded from http://bit.ly/2m8WenA, which shows how to tag network connection from different sections of the application that can help analyze data traffic volume from the app. Figure 6 shows an example of how it looks like using Android Device Monitor (inside Studio go to Tools → Android → Android Device Monitor):

As can be seen in the graph, the sample app has 3 different connections, and each one is accessing a different website. By tagging them with different codes, we are able to see the amount of data traffic each connection causes, and this will assist us to see if our application is using bandwidth efficiently.

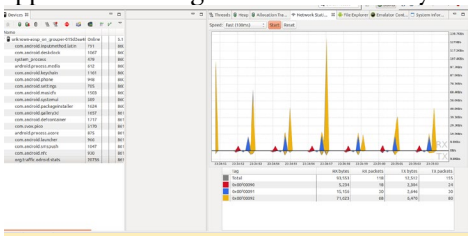


Figure 6 - Network Tag

The code in the following code snippet displays the usage of internal API TrafficStats. This API tag shows the thread that is currently using a socket with an id, and all traffic that goes through the same connection will be recorded together. This is how the app is able to group the 3 different connections together. Internally, Android uses the xt_qtaguid module in the kernel to do this type of tagging. There is more information about this available at http://bit.ly/2kU8KXB.

```
public void getUsingNormalURLC
onnection(String urlText, int
threadTag)
                throws IOException
{
    URL url = null;
    TrafficStats.
setThreadStatsTag(threadTag);
    . . . .
    HttpURLConnection urlConnec-
tion = getConnection(url);
    try {
       . . . . .
        }
    } finally {
        TrafficStats.clearThread-
StatsTag();
    }
}
```

SENSING THE PRESENCE

CHRONICLES OF A MAD SCIENTIST

by Bo Lechnowsky

Annoyed, you climb into your world domination 4x4, a vehicle which looks like an old farm truck from the outside, but inside looks more like the control system of a space shuttle, because the delivery driver at the oriental restaurant called in sick. Now you have to drive over to pick up your own food. "How primitive," you think to yourself as you back out of your secret garage. Just as you are about to bring the vehicle to a stop, you hear a loud crash! You get out and hurry around to the rear of the 4x4 where you see an obliterated trashcan. You pull what little hair you have left above your ears and think, "those Neanderthal garbage collectors left my can in the middle of my driveway again!" Now you have to pick up your dinner AND a new trash can.

As you drive off to run your errands, you consider how you can avoid this inconvenience in the future. Several thoughts come to mind:

Lidar system to 3D scan the surroundings in real time and sound alarms when necessary
Automated robot arm on the back of the 4x4 that uses video processing to sense objects, grab them and move them out of the way
A simple back-up camera

You consider the feasibility of each:

Super cool, but too complicated and expensive
Even more cool and a fitting project for a mad scientist, but too complicated and expensive
I already have two of those, but was too preoccupied in thought to pay attention to them!

What you need is something simple, uncomplicated and inexpensive that can sound an alarm to alert you, even when you are deep in thought. "Aha!" You remember seeing a new product at ameriDroid that can do just that. It's a USB-connected microcontroller that supports up to 6 ultrasonic sensors with simple serial commands (http://bit.ly/2l26ptV). "Perfect!" you think to yourself.

As you return to your subterranean laboratory, Kung Pao chicken in hand, you hurry over to your wall of monitors, pull up the ameriDroid website, and order a "USB Ultrasonic Ranging Sensor" kit with 6 sensors and cases.

You feel a sense of relief as you return to the 12 projects you were already working on while you wait a couple of days for your order to arrive. While eating your Kung Pao, you draw a diagram of how you want to arrange the sensors on the back of your vehicle.

A couple of days later, the kit arrives. You immediately set to assembling the kit and mounting the sensors on the back of your 4x4, being careful to use silicone sealant to protect the sensors from any rain, sleet and snow you may encounter on your world domination adventures.

Next, you refer to the instructions provided by ameriDroid with the kit on how to control and read the sensors from your trusty ODROID-C2 and VU7 that you mounted in the dash for just this purpose.

Connecting the USB cable

The supplied USB cable should be connected to the microcontroller (the small circuit board with the rows of pins sticking out of it). The microcontroller has a row of 4 or 6 pins on

one end. If it has 6 pins, concentrate on the 4 center pins and follow the directions below:

Black - Connect to GND
Red - Connect to VCC
Green - Connect to RXI
White - Connect to TXD

Connecting the ultrasonic sensors

On each ultrasonic sensor, there are four labeled pins: Vcc, Trig, Echo, Gnd. For the first ultrasonic sensor, connect pin 2 on the microcontroller to the "Trig" pin on the ultrasonic sensor. Connect pin 3 on the microcontroller to the "Echo" pin on the sensor. Connect the "Vcc" pin on the sensor to 5V DC (the "VCC" pin on the microcontroller supplies 5V DC), and the "Gnd" pin on the sensor to a ground connection, such as one of the "GND" pins on the microcontroller.

If you are connecting up a second ultrasonic sensor, connect "Trig" to pin 4 and "Echo" to pin 5. Continue connecting to the next higher pin numbers for additional sensors, up to pins 12 and 13 if you want to connect six ultrasonic sensors.

Software connection

Use a terminal program like PuTTY, Screen or your favorite programming or scripting language to connect to the serial port that appears when you connect the ultrasonic sensor. This will vary based on the operating system used to connect to the microcontroller. In Windows, "Device Manager" is the common way to find this out. In Linux, "dmesg" or "lsusb" are common ways to detect which port it is connected to. You may see something called "PL2303", which is the microcontroller connection. If you receive garbage in your terminal when you connect, make sure the serial settings are 9600 baud, 8 bits, none (parity) and 1 stop bit. Here is a list of the commands that are accepted by the microcontroller:

- debug on: Enables detailed feedback. This is disabled by default, and will provide some clairifcation on input errors.
- debug off: Disables detailed feedback.
- init x y: Initializes an HC-SR04 on the pins specified: X is trigger, Y is echo. These must be digital pins. Device 0 is preset to pins 2 and 3, so those pins do not need to be specified. Invalid pin selections will return an exclamation mark.
- ping: Outputs a single read of the previously used HC-SR04. If none have been used yet, it will use device 0. If the command is followed by a single space and a number between 0-100 (exclusive), that number of reads will be made, outputting first the average of those reads, then the number of failed reads (which do not count towards the average), then the minimum and maximum read values separated by spaces. In any case, if a signal times out, -1 will be returned.
- start: Constantly pings all connected devices in sequence. Because the devices are used one at a time, the flow of data from this command will be faster if the devices measure short distances. If this command is followed by a single space and a number greater than 0, the board will wait that many milliseconds between reading from the last device and reading from the first device. Output from this command will be each device's reading (with a single space after each), and a return after the last device.
- stop: stops the 'start' operation, which is only effective after "start has been issued.
- mode: outputs the current measuring mode: M for Metric (millimeters), I for Imperial (tenths of an inch), or R for Raw (the pulse length in microseconds returned by the HC-SR04). The default is M.
- timeout: outputs the current timeout for signal reads. Timeout is the longest amount of microseconds that the board will wait before declaring the HC-SR04 to have made a bad read, which is not necessarily the same thing as the longest pulse the board will accept from the HC-SR04. When the board measures the pulse from the HC-SR04, it first waits for the HC-SR04 to begin sending a pulse at all. This time counts towards the timeout. When this command is followed by a space and a number, timeout will be set to that number. The default is 1 second (1,000,000 microseconds).
- ver: outputs version information and credits.

If any command (or no command) is preceded by a single digit from 0-5 (inclusive) followed by a space, the device which corresponds to that digit will be selected for use with the next "ping" command. Each time the board is ready for a command, it will output ">" to the serial. The only exception is after the "start" command, which will not prompt for input until after "stop" has been recognized.

MEET AN ODROIDIAN

VIACHESLAV ALEKSEEV

edited by Rob Roy (@robroy)

Please tell us a little about yourself.

I'm a 47-year old electronics and software engineer from Russia. I was born deep in Siberia, and after I finished school, I moved to Moscow to study at the university. Being a student at the Moscow Aviation Institute (MAI) was an amazing experience, probably the best in my life. In the early 1990s, a microcomputer era came to Russia, and I used to play a long game of upgrade leaps starting from the z80 CPU to the i486 and beyond. After I finished my university studies and post graduate work, I became an engineer. I worked a few jobs at different companies, and eventually decided to start my own business. In 2006, I established a small startup for creating automobile traffic counting systems. I'm married and have a 21-year old daughter named Lena, who works as a nurse. My wife Nadezhda is a production dress designer, but she now works as web designer. She is fond of digital photography and likes her DSLR camera very much.

1995, at the MAI campus, testing one of the world's first commercial VR harnesses, which ran at 640x480 @ 30fps. It used a magnetic sensor at the nape for head positioning. The best prank to play on someone was to bring a magnet near the rear side and shake it. It guaranteed screams when the 3D world rolled around!

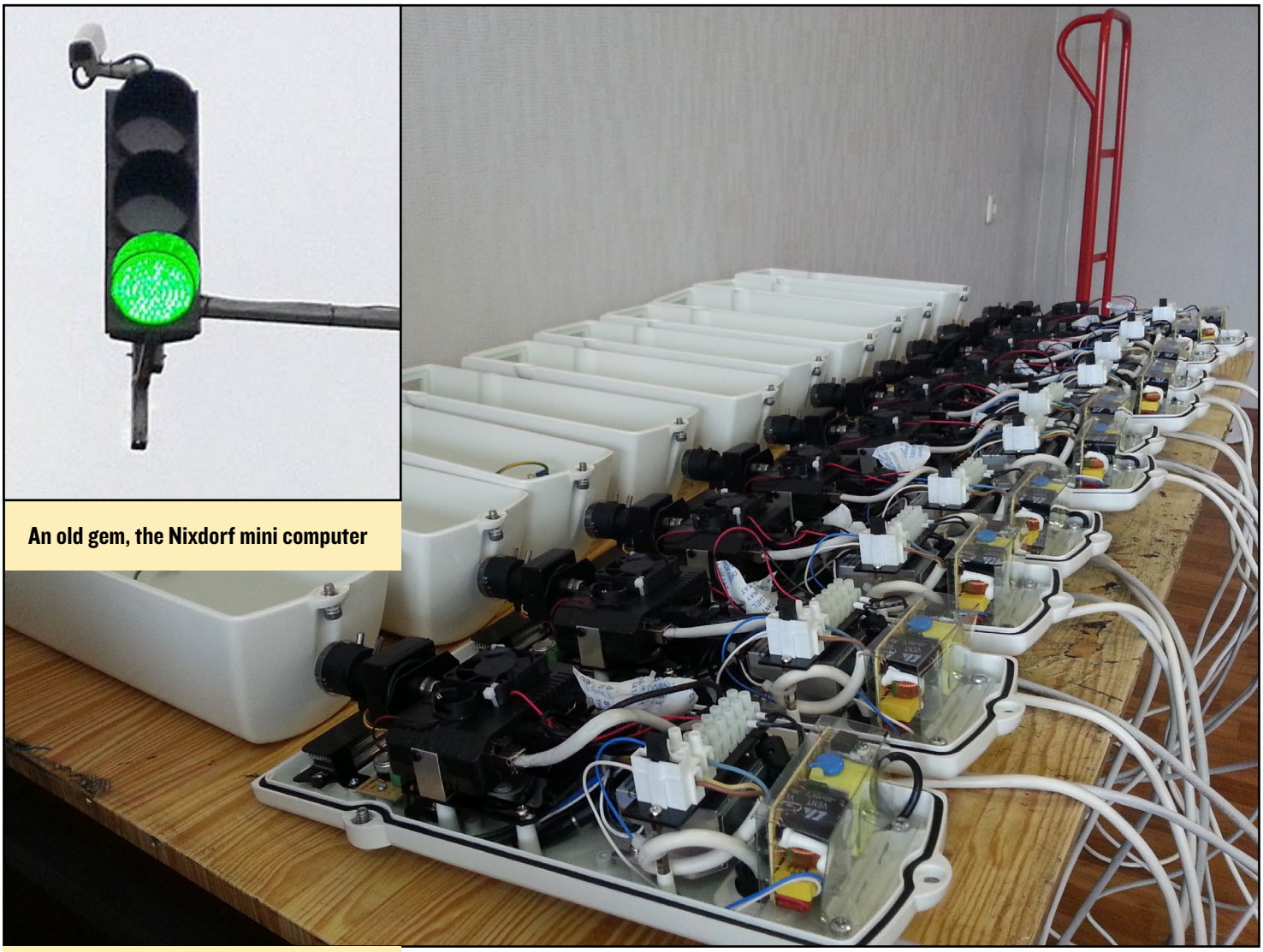
Viacheslav enjoys playing guitar and listening to music

How did you get started with computers?

When I started my business, I had to decide which hardware platform would be used for road traffic data acquisition. Just so you don't hate me, my traffic cameras are not those which are used to issue speeding tickets or anything like that. My system is dedicated to smoothing traffic flow by determining the optimal traffic signal controls. Previously, I used an industrial PC to run my software image recognition engine. I now use the ODROID family of devices for this task instead.

What attracted you to the ODROID platform?

In 2012, I used Google to search for a very compact but powerful platform for my needs, which was the



An old gem, the Nixdorf mini computer

An old gem, the Nixdorf mini computer

ODROID-X2 board. With its four cores running at 1.7 GHz, it easily computed automobile traffic counting algorithms. Later on, I switched to using the U3 and the XU4.

Which ODROID is your favorite and why?

At the moment, the XU4 is my favorite. My system is based on a real-time video frame processing, so the USB 3.0 bus on the XU4 is quite good as a camera video capture interface. For my system, it is very important to have a good bridge between the video camera and the CPU. I will probably have to look to the lower level interfaces like MIPI CSI-2, which is unfortunately absent on the XU4. Recently, I've been learning how to use the oCam GS BW camera. The camera is good by itself, but for industrial outdoor use, I have to implement a software automatic exposure control and probably lens aperture control, which is still under consideration.

What hobbies and interests do you have apart from computers?

I enjoy driving and reading about cars, and watching the F1 racing championships. I also like travelling, fishing, listening to music, and playing guitar. I have a Siberian cat named Leia, and am a Star Wars fan.

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

Making software is very cool and mystical. It's a fusion of art and technology. Nothing is more inspirational than when your application starts to live. However, there is always the other side of the Force. Be ready for endlessly educating yourself. Software development is one of the fastest changing activities. If you plan to stop making software in one or two years, you can stop learning now. Knowledge will come to be outdated in a couple of years or so. To be at the highest level, you must always run. It's similar to the song like "Run like Hell" by Pink Floyd. If you ready to live in such a way, you will be successful.