Volumio 2 • Android ADB Debug • Android navigation using IR remote

ODROID Magazine

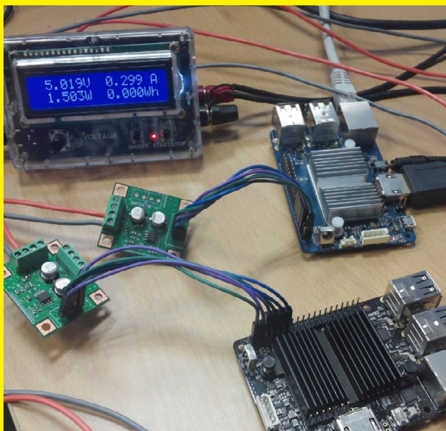
Year Four
Issue #41
May 2017

Repurpose your N64 With the power of ODROID

A complete walkthrough allowing you to use the classic Nintendo console case with your favorite board

Offering Native ODROID-C2 Support

Exploring RS485 communication on C1+ and C2



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

Elektronik, Technik und Sonderposten zu Superpreisen!

mein Konto | Kontakt & Service | Versand & Bezahlen | Katalog | Newsletter | Gutscheine

Merkliste | Neu registrieren | Anmelden

Direktbestellung

0 Artikel: 0,00 €
Einkaufswagen

Suchen nach: Suchbegriff oder Bestellnummer Los

Sie sind hier: Startseite > Suche

Suchergebnis Ihre Suche nach "odroid" ergab 18 Treffer

sortieren nach: Relevanz (gewählt)

1 - 18 von 18 Ergebnissen

Seite: 1 von

Kategorie wählen

Filtereinstellungen

[-] alle Filter entfernen

Kategorie
[+] Bausätze / Module (17)
[+] Stromversorgung (1)

Preis 1 € 80 €

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

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

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 Android
Superschnelles N den ODROID-U3 Betriebssystem aufgespielt un
39,95 €*
Best.Nr. 810

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- Linux
Supersch den OD Linux- berei
39

Do you have an old Nintendo or other gaming console that doesn't work anymore? Don't throw it away! You can refurbish it with an ODROID-XU4 running ODROID GameStation Turbo, RetroPie or Lakka and turn it into a multi-platform emulator station that can play thousands of different console games. Our main feature this month details how to fit everything into an N64 shell, breathing new life into an old dusty console case.

ODROIDs are extremely versatile, and can be used for music playback, as described in our Volumio 2 article, developing Android apps, as Nanik demonstrates in his article on the Android Debug Bridge, and process control, as shown by Charles and Neal in their discussion of the RS485 communication protocol. We also have a guide to setting up RetroPie, a gaming OS, which now offers native ODROID-C2 support in version 4.2. Adrian details how to add more buttons to your ODROID-C1 or C2, Lorenzo presents his technique for using an infrared remote control with Android, and Tobias continues his Linux Gaming column with a look at the very popular racing game series called F-Zero.

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 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

Went bonkers. Again!

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.

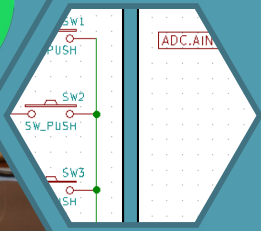
Andrew Ruggeri, Assistant Editor

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.

INDEX



VOLUMIO 2 PLUGINS

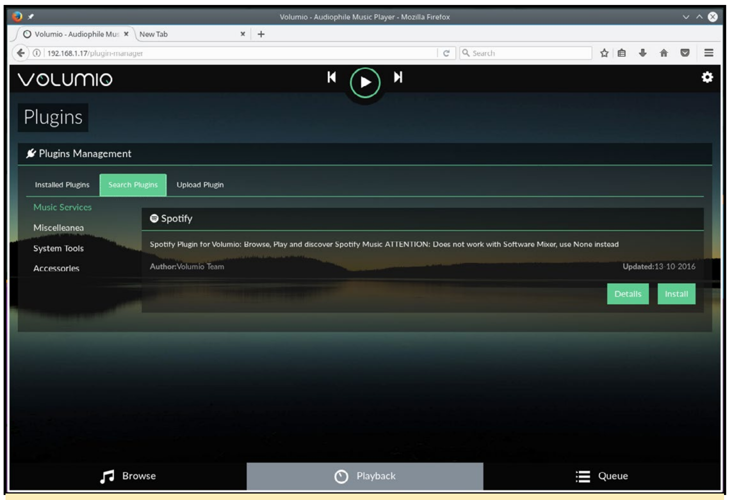
STREAM MUSIC WITH SPOTIFY

by @synportack24

If you're a fan of high fidelity music, or just love rocking out, Volumio 2 is a great feature-packed OS that is just what you're looking for. Almost a year ago, August to be exact, I wrote an article about the awesome new features that Volumio 2 brought to the ODROID family. One such feature was the ability for developers to easily add and create plugins. This article is intended to get you up and running with one of my favorite plugins called Spotify.

If you're not familiar with Volumio 2, or if you want to get the base OS up and running, I recommend having a look at the original article first, which is located on page 26 of the August 2016 issue of ODROID Magazine at http://bit.ly/2pUExh8

Installation



Volumio - Audiophile Music Player - Mozilla Firefox

192.168.1.17/plugin-manager

Plugins

Plugins Management

Installed Plugins | Search Plugins | Upload Plugin

Music Services
Miscelleanea
System Tools
Accessories

Spotify

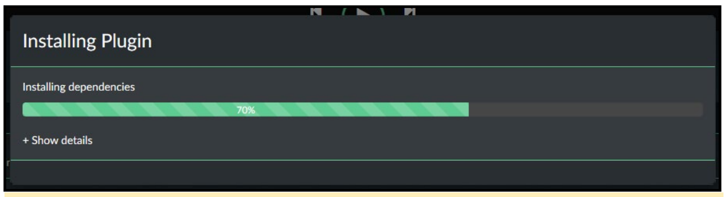
Spotify Plugin for Volumio: Browse, Play and discover Spotify Music ATTENTION: Does not work with Software Mixer, use None instead

Author:Volumio Team Updated:13-10-2016

Volumio 2 plugins page

Installation is very straight forward. Click on the gear icon in the upper right of the screen and select "Plugins" from the side menu. Once the plugins page loads, select the "Search Plugins" tab, followed by the "Music Services" filter on the left hand side, which should be the default category. On a clean version of Volumio 2, Spotify will be the only plugin listed under this filte, although more are available from 3rd party developers if you so desire. To install the plugin, simply click the "Install" button on the bottom right of the Spotify plugin box.

Once you click the install button, Volumio will show a popup box installer progress bar. For me, it took several minutes, since Volumio needs to download and install several required dependencies. After it installs, refresh the browser page, and you will now see the plugin listed under the "Installed Plugins" tab. Click the "on" toggle, then the settings button to enter in your Spotify login information, and save it. After this, you are ready to go!



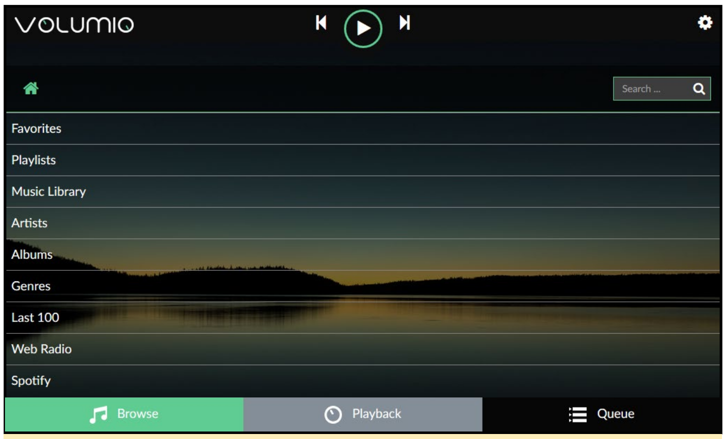
Installing Plugin

Installing dependencies

Installation progress of the Spotify plugin

Play

Now when you go the the playback options at the bottom of the list, you will see an option for "Spotify". Click this, and you can browse through your playlists, as well as all other stations normally found on Spotify.



Spotify playback option on bottom of the list

For more information please visit the Volumio 2 website at www.volumio.org.

MULTICLICK BUTTON HANDLER FOR 3.5" LCD AND WEBCAM

GETTING THE MOST OUT OF THE HARDWARE BUTTONS

by Adrian Popa

One thing I really liked about the ODROID 3.5" touchscreen module is the inclusion of 4 hardware switches that can be programmed to do just about anything. But there is one shortcoming in my opinion: I wish there were more buttons available. Can anything be done about that?

In order to get started using the 3.5" touchscreen, I found it best to follow @fourdee's guide at `http://bit.ly/2oC7mdw`. This will help set up the touchscreen, but it does not suggest any use for the buttons. While Hardkernel provides some sample code (`http://bit.ly/2pUogZy`) to convert the 4 buttons in a virtual keyboard that can output 4 key presses, we will not be using it in this article.

My first idea of getting more out of these buttons was to see if I could click 2 buttons simultaneously and generate a different event. The way the buttons work is to create a "short" between the ADC pin and ground through one or more resistors. Based on the resulting resistance value, the ADC produces a numerical value which helps you identify which button has been pressed.

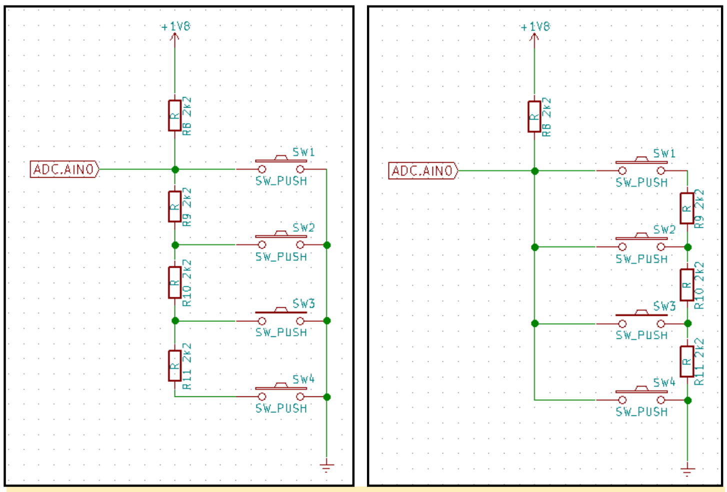


Figure 1 (a) The original HK design & 1 (b) improvement option

Figure 1a shows the design Hardkernel chose, while Figure 1b shows a small modification which could have added 3 more "events" from these buttons. Had the resistors been mounted closer to the ground, instead of before the switch, you could have created a parallel circuit when pressing two buttons and have a different ADC value. I did the math (I was surprised I still knew how 10 years after college) and worked out that for an ODROID-C1 or C2, the values shown in Figure 2 would have been possible with the reversed resistors.

Figure 2 - ADC values for multiplexed buttons (as table)

SW4	5 ±10
SW3	515 ±10
SW2	680 ±10
SW1	770 ±10
SW1+SW3	438 ±10
SW2+SW3	408 ±10
SW1+SW2	558 ±10

One possible issue with this approach might have been getting "misclicks" if the buttons were not pressed simultaneously, but since Hardkernel didn't implement the hardware this way, we need to explore other alternatives.

A different way to handle key events is to handle multiple key presses as a single event. For example, you could handle a "double/triple-click" event or even a "long-press" and have it execute something else. The way you can detect such an event is to set up a buffer, record key presses in the buffer, and when the buffer is full, process the event.

Since this seemed like a fun and simple project, my plan was to expand Hardkernel's C code that handled the keys and add this functionality. However, having been away from C programming for quite a while, I decided to do it in Perl.

The code works a bit like this: There is an infinite loop reading the ADC input. Once a change has been detected (due to a key press), the value is recorded in a buffer. The subsequent reads are also stored in the buffer (even when no key is pressed) and when the buffer is full, the key sequence is identified, and some action is taken. This approach had the fortunate side-effect that it adds support for key sequences such as

"KEY1-KEY2" and "KEY1-KEY2-KEY3".

To get the code and set it up, follow the steps below:

```
$ git clone https://github.com/mad-ady/tftlcd35-key.
git
$ cd tftlcd35-key
$ sudo apt-get install liblog-log4perl-perl \
libproc-background-perl libconfig-yaml-perl
$ sudo cp tftlcd35_key.pl /usr/local/bin
$ sudo cp tftlcd35-key.service /etc/systemd/system/
```

The code ships with a few configuration examples to use based on your needs. If you only want 1 normal click and 1 long-click per key, use config-empty-1.yaml. If you want to use 2 key combinations use config-empty-2.yaml or if you want 3 key combinations use config-empty-3.yaml. The configuration syntax is in simple YAML format. Just as in Python syntax, indentation counts, and you should not use the tab character to create white space. If, after starting the script, it generates complaints about the configuration, you can use an online validator like `http://www.yamllint.com/` to pinpoint the problem. The default configuration sets logging to INFO level, which sets the period between reads to 200ms, the buffer length to 10 and the longpress interval to 70% of the buffer length. If you have issues, you can set the logging to DEBUG level, but it's very verbose. To understand how these parameters affect you, let's analyze the hypothetical buffer shown in Figure 3.

KEY1	KEY1		KEY2	KEY2	KEY2		KEY3		

Figure 3 - Sample buffer

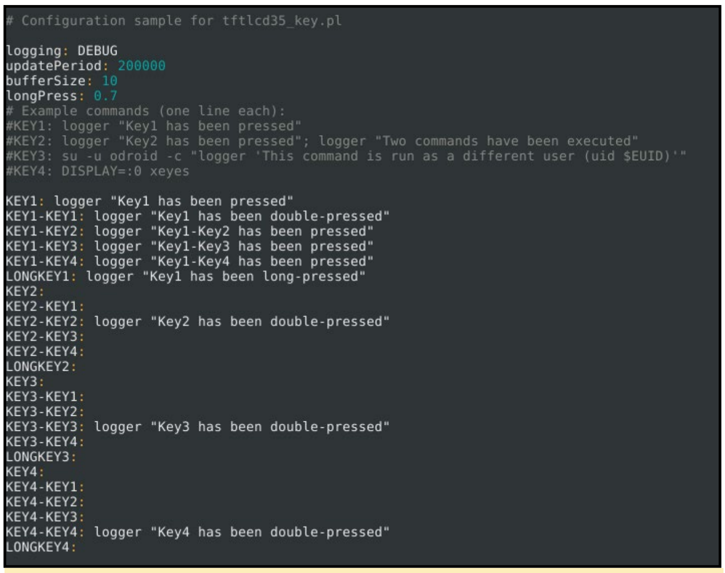
Each cell represents a value read from the ADC. The buffer has 10 cells, so bufferSize has to be 10. Depending on how fast you press and release a key, you can set the updatePeriod higher or lower (default is 200000 microseconds, or 200ms). If you set updatePeriod to something too long (e.g. 500ms), you will miss keypresses because you can press and release the button when the script is in sleep mode. If you set updatePeriod to something too short, you may need a bigger buffer to record all of your key presses. For instance, if you get the buffer listed earlier in Figure 3, but you double-clicked KEY1, this means your updatePeriod is too high.

From the moment you press a key, the script will take updatePeriod * bufferSize microseconds to react. This means that having a high bufferSize gives you a high reaction time (you pressed the button once, but the action happens 5 seconds later). The default configuration uses a 2 second reaction time (200ms * 10).

The final global parameter is longPress which represents the number of items in the buffer that have to be a certain key before the buffer is interpreted like a long press. The default is 0.7, which means that 70% of the buffer has to be filled with a key press (for our example it would mean holding down a key for at least 1.4s). Note that if you hold down the key for too long (e.g. 2.2s) and the buffer size is exceeded, the program will interpret the first 2s as a long press and generate the desired event and the final 0.2s as a short key press and fire a different event. This can be mitigated in the code with a short sleep after a long event if needed.

The final part of the configuration is the key to command mapping. Key sequences are separated by a dash (-) and long presses are prefixed by LONG. In the example above the key sequence interpreted by the script is KEY1-KEY2-KEY3 (duplicated keys are ignored). To get a KEY1-KEY1-KEY1 sequence you need to release KEY1 for at least an updatePeriod so that the buffer contains a blank reading between two keys.

To map a command to a sequence, simply type in the command you want executed on the same line after the ':' sign. The commands are executed in a background shell as the same user as the script runs (root). If you need to run a graphical command, you can prefix it with DISPLAY=:0.



```
# Configuration sample for tftlcd35_key.pl

logging: DEBUG
updatePeriod: 200000
bufferSize: 10
longPress: 0.7
# Example commands (one line each):
#KEY1: logger "Key1 has been pressed"
#KEY2: logger "Key2 has been pressed"; logger "Two commands have been executed"
#KEY3: su -u odroid -c "logger 'This command is run as a different user (uid $EUID)'"
#KEY4: DISPLAY=:0 xeyes

KEY1: logger "Key1 has been pressed"
KEY1-KEY1: logger "Key1 has been double-pressed"
KEY1-KEY2: logger "Key1-Key2 has been pressed"
KEY1-KEY3: logger "Key1-Key3 has been pressed"
KEY1-KEY4: logger "Key1-Key4 has been pressed"
LONGKEY1: logger "Key1 has been long-pressed"
KEY2:
KEY2-KEY1:
KEY2-KEY2: logger "Key2 has been double-pressed"
KEY2-KEY3:
KEY2-KEY4:
LONGKEY2:
KEY3:
KEY3-KEY1:
KEY3-KEY2:
KEY3-KEY3: logger "Key3 has been double-pressed"
KEY3-KEY4:
LONGKEY3:
KEY4:
KEY4-KEY1:
KEY4-KEY2:
KEY4-KEY3:
KEY4-KEY4: logger "Key4 has been double-pressed"
LONGKEY4:
```

Figure 4 - Example configuration

Once the configuration is finished, you can copy it as /etc/tftlcd35-key.yaml, activate the script and have it start up automatically:

```
$ sudo cp custom-config.yaml /etc/tftlcd35-key.yaml
$ sudo systemctl enable tftlcd35-key
$ sudo systemctl start tftlcd35-key
```

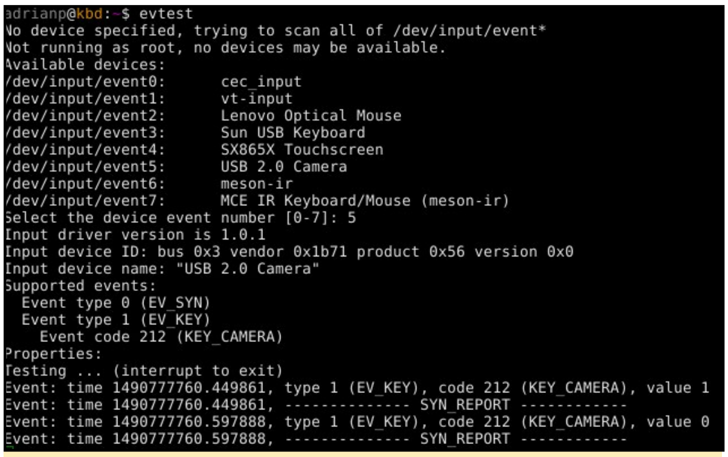
Logging and debugging information goes to syslog and can be viewed with the following command:

```
$ sudo journalctl -f -u tftlcd35-key
```

Using a button on a device

If you do not have the Hardkernel 3.5" display to use, but you have devices with physical buttons (like, a camera or a sound card) you can still get multiple actions from them in a similar way. For example, the Hardkernel 720p camera comes with a button to take snapshots, which is rarely used. The button registers as an input device (e.g., as a standard keyboard) and registers only 1 button as an event. You can easily find out which keys are supported by running the evtest command, as shown in Figure 5.

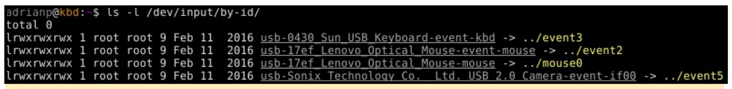
```
$ sudo apt-get install evtest
```



```
adrianp@kbd:~$ evtest
No device specified, trying to scan all of /dev/input/event*
Not running as root, no devices may be available.
Available devices:
/dev/input/event0:      cec_input
/dev/input/event1:      vt-input
/dev/input/event2:      Lenovo Optical Mouse
/dev/input/event3:      Sun USB Keyboard
/dev/input/event4:      SX865X Touchscreen
/dev/input/event5:      USB 2.0 Camera
/dev/input/event6:      meson-ir
/dev/input/event7:      MCE IR Keyboard/Mouse (meson-ir)
Select the device event number [0-7]: 5
Input driver version is 1.0.1
Input device ID: bus 0x3 vendor 0x1b71 product 0x56 version 0x0
Input device name: "USB 2.0 Camera"
Supported events:
  Event type 0 (EV_SYN)
  Event type 1 (EV_KEY)
    Event code 212 (KEY_CAMERA)
Properties:
Testing ... (interrupt to exit)
Event: time 1490777760.449861, type 1 (EV_KEY), code 212 (KEY_CAMERA), value 1
Event: time 1490777760.449861, -------------- SYN_REPORT ------------
Event: time 1490777760.597888, type 1 (EV_KEY), code 212 (KEY_CAMERA), value 0
Event: time 1490777760.597888, -------------- SYN_REPORT ------------
```

Figure 5 - evtest sample output

The input devices are mapped in /dev/input/event*, but the mapping is dynamic so you cannot depend on the numbers across reboots. Instead you should identify your device from /dev/input/by-id/ which is more stable, as shown in Figure 6.



```
adrianp@kbd:~$ ls -l /dev/input/by-id/
total 0
lrwxrwxrwx 1 root root 9 Feb 11  2016 usb-0430_Sun_USB_Keyboard-event-kbd -> ../event3
lrwxrwxrwx 1 root root 9 Feb 11  2016 usb-17ef_Lenovo_Optical_Mouse-event-mouse -> ../event2
lrwxrwxrwx 1 root root 9 Feb 11  2016 usb-17ef_Lenovo_Optical_Mouse-mouse -> ../mouse0
lrwxrwxrwx 1 root root 9 Feb 11  2016 usb-Sonix_Technology_Co.__Ltd._USB_2.0_Camera-event-if00 -> ../event5
```

Figure 6 - Stable input mapping

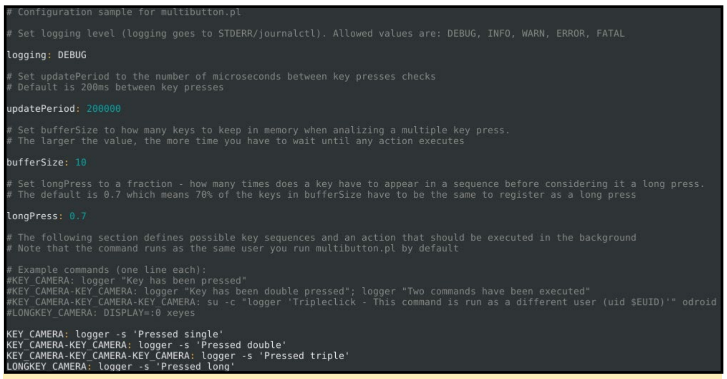
So, to benefit from click and multi-click events (double/triple/long-press) you can download and install this handler program:

```
$ git clone https://github.com/mad-ady/multibutton.
git
$ cd multibutton
$ sudo perl -MCPAN -e 'install Linux::Input'
$ sudo apt-get install libconfig-yaml-perl liblog-log-
4perl-perl \
libproc-background-perl
$ sudo cp multibutton.pl /usr/local/bin
$ sudo cp config-minimal.yaml /etc/multibutton.yaml
$ sudo cp multibutton.service /etc/systemd/system
```

Before starting the service, you will need to configure it. The first step is to identify your input device and provide the correct path to it in /etc/systemd/system/multibutton.service (the -i parameter), as described above.

Next, adjust the configuration file to meet your your needs. The syntax is the same as the touchscreen handler, however, since in my case I have only 1 button, there is only one thing I can control. The configuration file is in YAML format and has the same 3 configuration options to change: polling period, buffer size and long press percentage. In this case, updatePeriod is not as important, since the kernel will do the polling and will forward all events to the end user (no more missed keys).

Then comes the key sequences in the format "KEY_LABEL1-KEY_LABEL2". You can get the correct labels with the evtest command. Following the key labels you need to specify the command(s) to be run. Figure 7 shows an example (see the actual file for comments):



```
# Configuration sample for multibutton.pl

# Set logging level (logging goes to STDERR/journalctl). Allowed values are: DEBUG, INFO, WARN, ERROR, FATAL

logging: DEBUG

# Set updatePeriod to the number of microseconds between key presses checks
# Default is 200ms between key presses

updatePeriod: 200000

# Set bufferSize to how many keys to keep in memory when analizing a multiple key press.
# The larger the value, the more time you have to wait until any action executes

bufferSize: 10

# Set longPress to a fraction - how many times does a key have to appear in a sequence before considering it a long press.
# The default is 0.7 which means 70% of the keys in bufferSize have to be the same to register as a long press

longPress: 0.7

# The following section defines possible key sequences and an action that should be executed in the background
# Note that the command runs as the same user you run multibutton.pl by default

# Example commands (one line each):
#KEY_CAMERA: logger "Key has been pressed"
#KEY_CAMERA-KEY_CAMERA: logger "Key has been double pressed"; logger "Two commands have been executed"
#KEY_CAMERA-KEY_CAMERA-KEY_CAMERA: su -c "logger 'Tripleclick - This command is run as a different user (uid $EUID)'" odroid
#LONGKEY_CAMERA: DISPLAY=:0 xeyes

KEY_CAMERA: logger -s 'Pressed single'
KEY_CAMERA-KEY_CAMERA: logger -s 'Pressed double'
KEY_CAMERA-KEY_CAMERA-KEY_CAMERA: logger -s 'Pressed triple'
LONGKEY_CAMERA: logger -s 'Pressed long'
```

Figure 7 - multibutton.pl sample configuration

Once you have set up your configuration and systemd service you can turn the service on with:

```
$ sudo systemctl daemon-reload
$ sudo systemctl enable multibutton
$ sudo systemctl start multibutton
```

You can view the logs with journalctl:

```
$ sudo journalctl -u multibutton -f
```

If you need to support multiple devices (e.g., say 2 cameras) you can create a new service, point it to the new device and load a different configuration.

Note that you can run this event handler on any input device, including a real keyboard, and only configure certain key presses to execute various commands. This could make for an interesting alternative keystrokes where, for example, you bind "w" to execute some arbitrary command when pressed three times in a row. Note that the code has not been tested for handling multiple keys.

For comments, questions, suggestions, or bug reports, please visit the support thread at http://bit.ly/2nFxvJh.

XU4 GAMING EMULATOR

TIME TO REPURPOSE THAT OLD CONSOLE YOU BOUGHT FROM THE FLEA MARKET

by @MarkT

I created an N64 gaming emulator with which I can play games from Atari all the way up to N64 and PS1. I modeled all the parts myself to fit into the case of an old, unusable N64 system. I am using an ODROID-XU4, which makes it much easier and smoother for 4 players at once compared to a Raspberry Pi 3.

I am using RecalBox as my emulation OS, which works great, although it has a poor controller configuration options when it comes to Nintendo 64. When building it, a hot glue gun was definitely my best friend to be sure that my connections were tight, and my USB ports were secure. I am still working on the routing of the HDMI, so if anyone else would like to put their own spin on it, I would be more then happy to see what you have in mind! This is just a rough draft of my project so far, and I will make a much more detailed procedure regarding the steps that I took once I have it more complete. Please do not do this to a working Nintendo 64 system! It broke my heart enough for me to gut this non-working system.

Materials

- A old Nintendo 64 system (preferably broken) and its hardware (screws)
- ODROID-XU4 W/ power supply
- Slim 4-Port USB Data Hub like this one, such as http://amzn.to/2papHjz
- 2 x 1/2' 2-56 screws
- SPDT MOM-ON Switch, such as http://bit.ly/2pYldPS
- 2 x 1/4' 4-40 Screws with nuts
- 1/2' 4-40 screw
- 28 AWG wire
- Soldering Iron
- Hot glue gun

Blue LED with a 10 Ohm Resistor

RJ-45 Jack similar in size to the one at http://bit.ly/2ooqkZM

A power jack similar size (9mm x 15mm) to the one at http://bit.ly/2oEhoe4

- I used Retrolink USB controllers, which gives you the feeling of playing the original system, but you can also use almost any USB controller, such as Xbox or PS3

RecalBox/Batocera OS (http://bit.ly/2ooDtSz)

Lakka OS (http://bit.ly/2ptdWHX)

3D Printing

The 3D print files may be downloaded from the Thingiverse project page at http://bit.ly/2pXcaf6. Flip the USB ports on their back (outlets pointing up) with rafts and support, then print 2 game pack hold downs. Use raft and support for both buttons/switches and the expansion pack cover.

Manufacturer:	MakerBot
Printer:	MakerBot Replicator
Rafts:	Yes
Supports:	Yes
Resolution:	.20
Infill:	30%

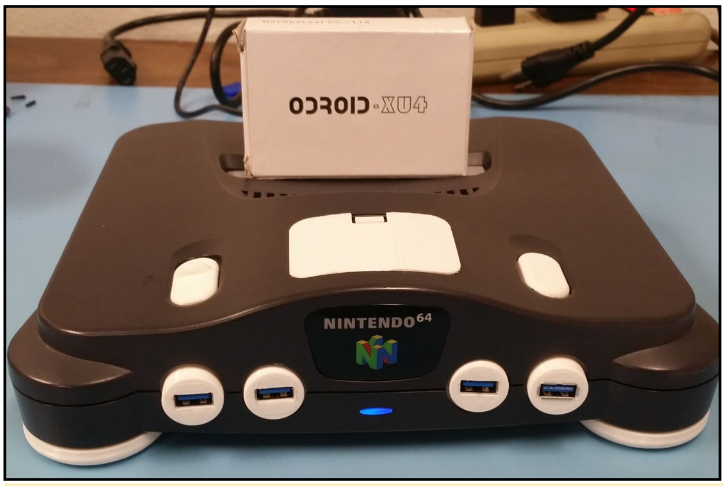


Figure 1 - 3D printed case, fully assembled

As shown in Figure 2, everything is placed as it should be. I have used hot glue to hold the USB terminals securely in the slots, and zip ties for strain relief. My USB hub sits under the raft. I have also attached an LED with a 10 Ohm resistor onto GPIO 11 and GND. I soldered the other end of my switch to the same switch that is on the board to the correct terminals.

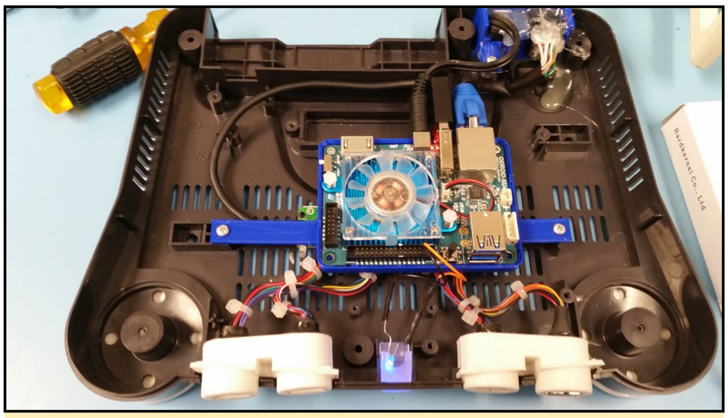


Figure 2 - Inside of case

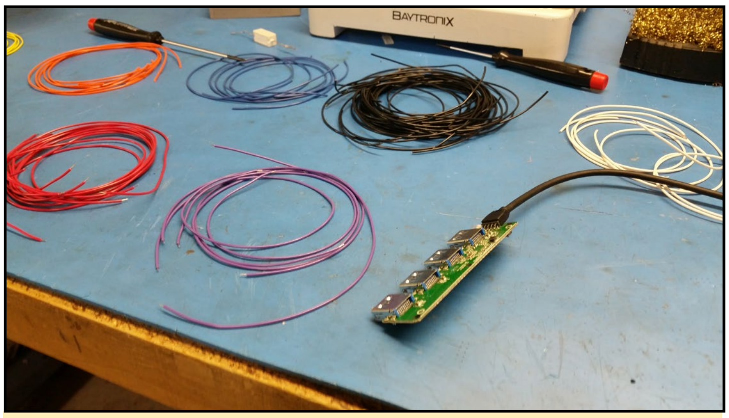


Figure 5 - Desolder the USB Ports and use roughly 28-30 AWG wire to extend the USB ports out to the printed USB slots. or buy new USB terminals, which are easier to solder

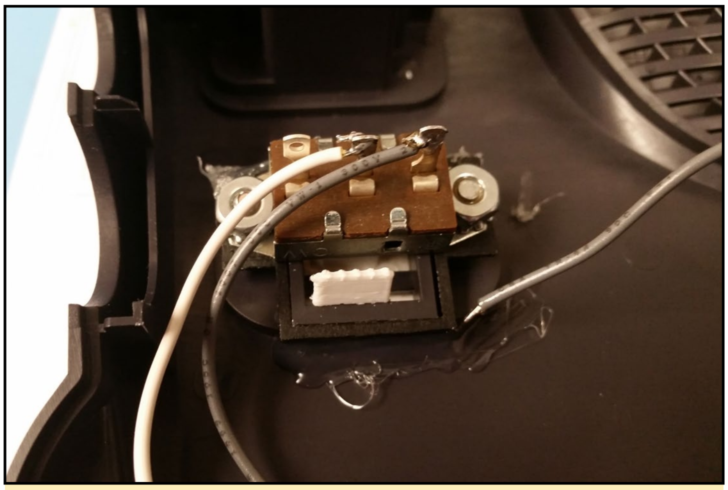


Figure 3 - The switch is screwed to the Switch_bottom print and glued in place to secure then, then the white and grey wire are soldered to the switch on the ODROID board.

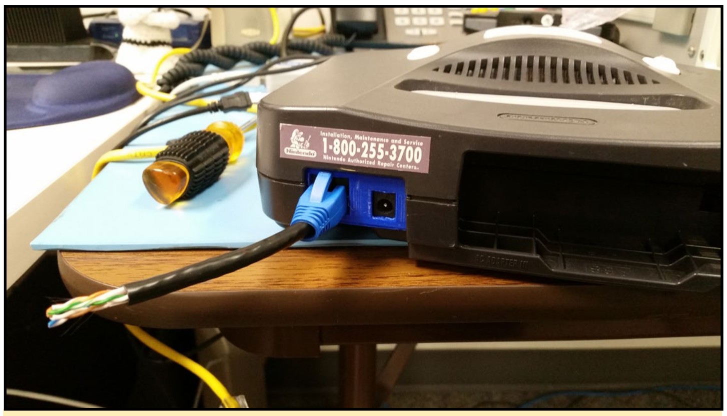


Figure 6 - The rear of the console showing the Ethernet outlet and DC power jack.

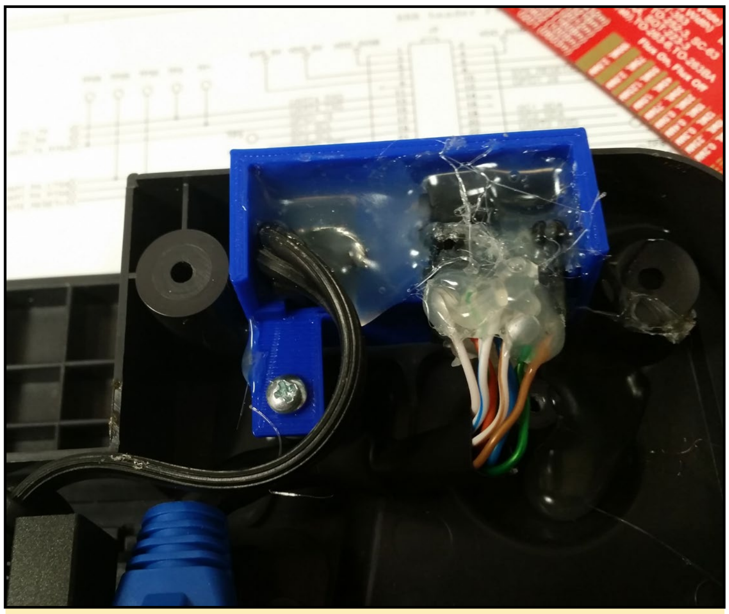


Figure 4 - Ethernet and DC jack are soldered and glued securely, making sure to have the DC jack soldered to the correct polarization before gluing.

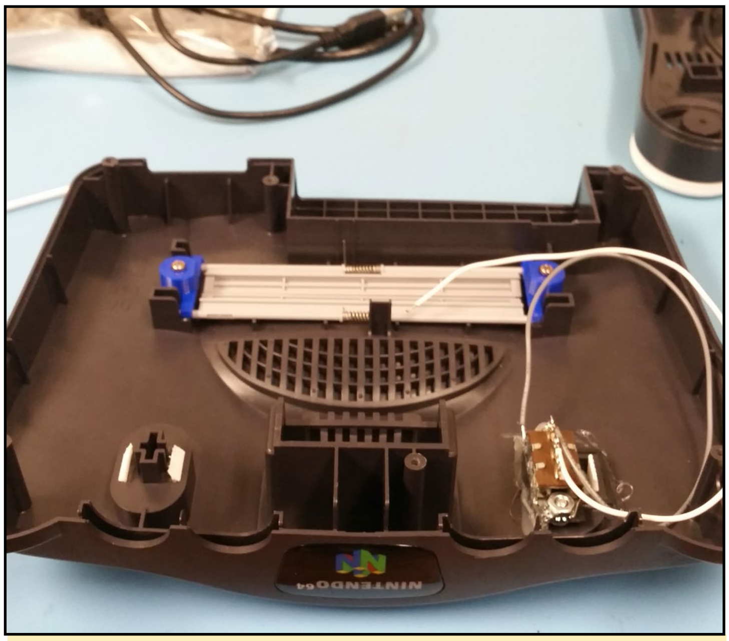


Figure 7 - For the top, I printed the game pack hold downs to make more space inside the console while maintaining the appearance of the Nintendo 64

DEEP DIVING INSIDE ANDROID

DEBUG BRIDGE (ADB) – PART 1

by Nanik Tolaram

Android Debug Bridge (ADB) is the only tool available for developers to communicate with their Android devices. ADB provides the ability for developers to dig deeper into Android by allowing them access to the command line shell, which opens the whole world of Android at their fingertips. Having access to the command line shell, we can take a look at the logs, pull data out from Android devices, installing and uninstall applications, and much more.

We are going to take a look at ADB in depth and dissect how it works internally. The article will be broken down into 2 part. This article is the the first part of the series, and will talk about ADB in general, and how it is being built as part of the build process and it's architecture. The second part will look more in detail at the different parts of the ADB source code and how the magic of communication with your device works.

ADB in L, M and N

Let's take a look first how different the adb source code is for the different version of Android. Figures 1, 2 and 3 show the source code for Lollipop, Marshmallow, and Nougat respectively. The location of the adb application is inside <your_android_directory>/system/core/adb folder.

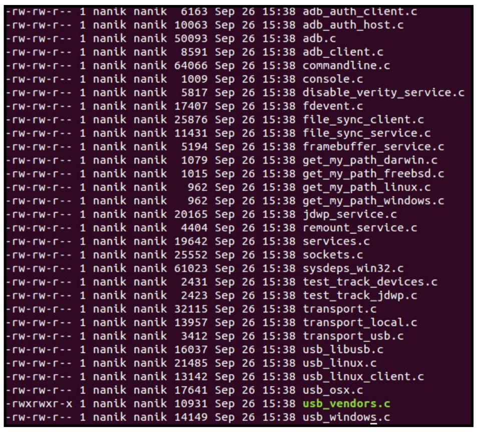
From the screenshot, you can see that ADB in Lollipop was written for C, while the other 2 versions are written in C++. The number of files have increased in Nougat because of the new functionality added, and a lot of refactoring has been done from previous versions to make the code more structured.

In terms of compilation artifacts, there are mainly 2 binary that we are interested in: adb and adbd, as shown in next page.

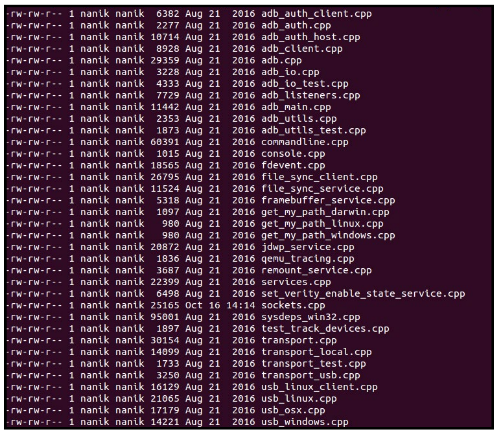
ADB Architecture

Internally, ADB is structured as 2 different applications that run as host and server. The "host" runs on your computer or laptop, and the "server" runs on the Android device. The applications run as a pair of client server connecting with each other either by IP or USB. Once connection has been established, commands are sent back and forth.

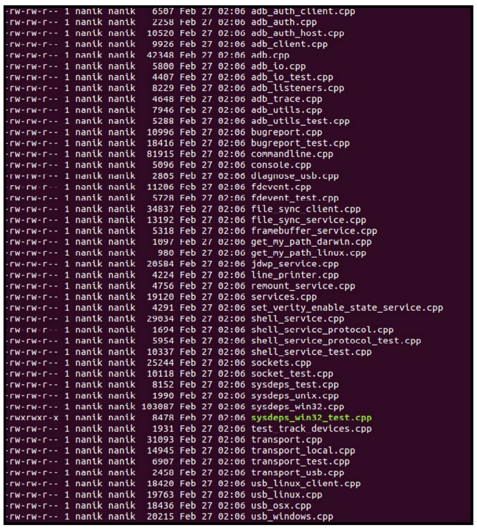
Figures 1, 2 and 3 - from left to right: Lollipop/Marshmallow/Nougat ADB source.



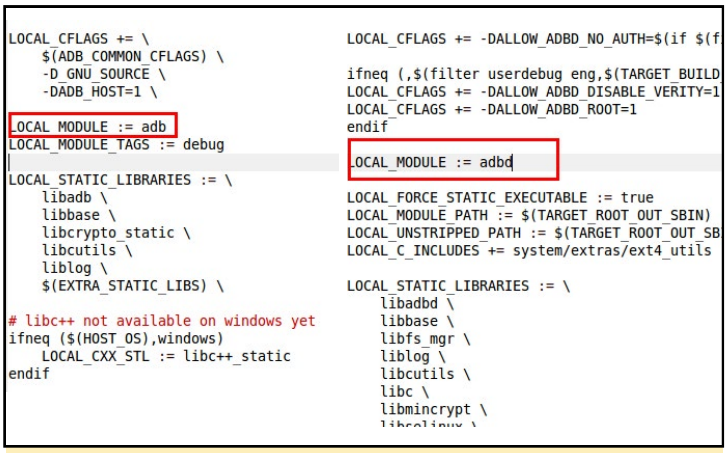
```
-rw-rw-r-- 1 nanik nanik  6163 Sep 26 15:38 adb_auth_client.c
-rw-rw-r-- 1 nanik nanik 10063 Sep 26 15:38 adb_auth_host.c
-rw-rw-r-- 1 nanik nanik 50093 Sep 26 15:38 adb.c
-rw-rw-r-- 1 nanik nanik  8591 Sep 26 15:38 adb_client.c
-rw-rw-r-- 1 nanik nanik 64066 Sep 26 15:38 commandline.c
-rw-rw-r-- 1 nanik nanik  1009 Sep 26 15:38 console.c
-rw-rw-r-- 1 nanik nanik  5817 Sep 26 15:38 disable_verity_service.c
-rw-rw-r-- 1 nanik nanik 17407 Sep 26 15:38 fdevent.c
-rw-rw-r-- 1 nanik nanik 25876 Sep 26 15:38 file_sync_client.c
-rw-rw-r-- 1 nanik nanik 11431 Sep 26 15:38 file_sync_service.c
-rw-rw-r-- 1 nanik nanik  5194 Sep 26 15:38 framebuffer_service.c
-rw-rw-r-- 1 nanik nanik  1079 Sep 26 15:38 get_my_path_darwin.c
-rw-rw-r-- 1 nanik nanik  1015 Sep 26 15:38 get_my_path_freebsd.c
-rw-rw-r-- 1 nanik nanik   962 Sep 26 15:38 get_my_path_linux.c
-rw-rw-r-- 1 nanik nanik   962 Sep 26 15:38 get_my_path_windows.c
-rw-rw-r-- 1 nanik nanik 20165 Sep 26 15:38 jdwp_service.c
-rw-rw-r-- 1 nanik nanik  4404 Sep 26 15:38 remount_service.c
-rw-rw-r-- 1 nanik nanik 19642 Sep 26 15:38 services.c
-rw-rw-r-- 1 nanik nanik 25552 Sep 26 15:38 sockets.c
-rw-rw-r-- 1 nanik nanik 61023 Sep 26 15:38 sysdeps_win32.c
-rw-rw-r-- 1 nanik nanik  2431 Sep 26 15:38 test_track_devices.c
-rw-rw-r-- 1 nanik nanik  2423 Sep 26 15:38 test_track_jdwp.c
-rw-rw-r-- 1 nanik nanik 32115 Sep 26 15:38 transport.c
-rw-rw-r-- 1 nanik nanik 13957 Sep 26 15:38 transport_local.c
-rw-rw-r-- 1 nanik nanik  3412 Sep 26 15:38 transport_usb.c
-rw-rw-r-- 1 nanik nanik 16037 Sep 26 15:38 usb_libusb.c
-rw-rw-r-- 1 nanik nanik 21485 Sep 26 15:38 usb_linux.c
-rw-rw-r-- 1 nanik nanik 13142 Sep 26 15:38 usb_linux_client.c
-rw-rw-r-- 1 nanik nanik 17641 Sep 26 15:38 usb_osx.c
-rwxrwxr-x 1 nanik nanik 10931 Sep 26 15:38 usb_vendors.c
-rw-rw-r-- 1 nanik nanik 14149 Sep 26 15:38 usb_windows.c
```



```
-rw-rw-r-- 1 nanik nanik  6382 Aug 21  2016 adb_auth_client.cpp
-rw-rw-r-- 1 nanik nanik  2277 Aug 21  2016 adb_auth.cpp
-rw-rw-r-- 1 nanik nanik 10714 Aug 21  2016 adb_auth_host.cpp
-rw-rw-r-- 1 nanik nanik  8928 Aug 21  2016 adb_client.cpp
-rw-rw-r-- 1 nanik nanik 29359 Aug 21  2016 adb.cpp
-rw-rw-r-- 1 nanik nanik  3228 Aug 21  2016 adb_io.cpp
-rw-rw-r-- 1 nanik nanik  4333 Aug 21  2016 adb_io_test.cpp
-rw-rw-r-- 1 nanik nanik  7729 Aug 21  2016 adb_listeners.cpp
-rw-rw-r-- 1 nanik nanik 11442 Aug 21  2016 adb_main.cpp
-rw-rw-r-- 1 nanik nanik  2353 Aug 21  2016 adb_utils.cpp
-rw-rw-r-- 1 nanik nanik  1873 Aug 21  2016 adb_utils_test.cpp
-rw-rw-r-- 1 nanik nanik 60391 Aug 21  2016 commandline.cpp
-rw-rw-r-- 1 nanik nanik  1015 Aug 21  2016 console.cpp
-rw-rw-r-- 1 nanik nanik 18565 Aug 21  2016 fdevent.cpp
-rw-rw-r-- 1 nanik nanik 26795 Aug 21  2016 file_sync_client.cpp
-rw-rw-r-- 1 nanik nanik 11524 Aug 21  2016 file_sync_service.cpp
-rw-rw-r-- 1 nanik nanik  5318 Aug 21  2016 framebuffer_service.cpp
-rw-rw-r-- 1 nanik nanik  1097 Aug 21  2016 get_my_path_darwin.cpp
-rw-rw-r-- 1 nanik nanik   980 Aug 21  2016 get_my_path_linux.cpp
-rw-rw-r-- 1 nanik nanik   980 Aug 21  2016 get_my_path_windows.cpp
-rw-rw-r-- 1 nanik nanik 20872 Aug 21  2016 jdwp_service.cpp
-rw-rw-r-- 1 nanik nanik  1836 Aug 21  2016 qemu_tracing.cpp
-rw-rw-r-- 1 nanik nanik  3687 Aug 21  2016 remount_service.cpp
-rw-rw-r-- 1 nanik nanik 22399 Aug 21  2016 services.cpp
-rw-rw-r-- 1 nanik nanik  6498 Aug 21  2016 set_verity_enable_state_service.cpp
-rw-rw-r-- 1 nanik nanik 25165 Oct 16 14:14 sockets.cpp
-rw-rw-r-- 1 nanik nanik 95001 Aug 21  2016 sysdeps_win32.cpp
-rw-rw-r-- 1 nanik nanik  1897 Aug 21  2016 test_track_devices.cpp
-rw-rw-r-- 1 nanik nanik 30154 Aug 21  2016 transport.cpp
-rw-rw-r-- 1 nanik nanik 14099 Aug 21  2016 transport_local.cpp
-rw-rw-r-- 1 nanik nanik  1733 Aug 21  2016 transport_test.cpp
-rw-rw-r-- 1 nanik nanik  3250 Aug 21  2016 transport_usb.cpp
-rw-rw-r-- 1 nanik nanik 16129 Aug 21  2016 usb_linux_client.cpp
-rw-rw-r-- 1 nanik nanik 21065 Aug 21  2016 usb_linux.cpp
-rw-rw-r-- 1 nanik nanik 17179 Aug 21  2016 usb_osx.cpp
-rw-rw-r-- 1 nanik nanik 14221 Aug 21  2016 usb_windows.cpp
```



```
-rw-rw-r-- 1 nanik nanik   6507 Feb 27 02:06 adb_auth_client.cpp
-rw-rw-r-- 1 nanik nanik   2258 Feb 27 02:06 adb_auth.cpp
-rw-rw-r-- 1 nanik nanik  10520 Feb 27 02:06 adb_auth_host.cpp
-rw-rw-r-- 1 nanik nanik   9926 Feb 27 02:06 adb_client.cpp
-rw-rw-r-- 1 nanik nanik  42348 Feb 27 02:06 adb.cpp
-rw-rw-r-- 1 nanik nanik   5800 Feb 27 02:06 adb_io.cpp
-rw-rw-r-- 1 nanik nanik   4407 Feb 27 02:06 adb_io_test.cpp
-rw-rw-r-- 1 nanik nanik   8229 Feb 27 02:06 adb_listeners.cpp
-rw-rw-r-- 1 nanik nanik   4648 Feb 27 02:06 adb_trace.cpp
-rw-rw-r-- 1 nanik nanik   7946 Feb 27 02:06 adb_utils.cpp
-rw-rw-r-- 1 nanik nanik   5288 Feb 27 02:06 adb_utils_test.cpp
-rw-rw-r-- 1 nanik nanik  10996 Feb 27 02:06 bugreport.cpp
-rw-rw-r-- 1 nanik nanik  18416 Feb 27 02:06 bugreport_test.cpp
-rw-rw-r-- 1 nanik nanik  81915 Feb 27 02:06 commandline.cpp
-rw-rw-r-- 1 nanik nanik   5096 Feb 27 02:06 console.cpp
-rw-rw-r-- 1 nanik nanik   2805 Feb 27 02:06 diagnose_usb.cpp
-rw-rw-r-- 1 nanik nanik  11206 Feb 27 02:06 fdevent.cpp
-rw-rw-r-- 1 nanik nanik   5728 Feb 27 02:06 fdevent_test.cpp
-rw-rw-r-- 1 nanik nanik  34837 Feb 27 02:06 file_sync_client.cpp
-rw-rw-r-- 1 nanik nanik  13192 Feb 27 02:06 file_sync_service.cpp
-rw-rw-r-- 1 nanik nanik   5318 Feb 27 02:06 framebuffer_service.cpp
-rw-rw-r-- 1 nanik nanik   1097 Feb 27 02:06 get_my_path_darwin.cpp
-rw-rw-r-- 1 nanik nanik    980 Feb 27 02:06 get_my_path_linux.cpp
-rw-rw-r-- 1 nanik nanik  20584 Feb 27 02:06 jdwp_service.cpp
-rw-rw-r-- 1 nanik nanik   4224 Feb 27 02:06 line_printer.cpp
-rw-rw-r-- 1 nanik nanik   4756 Feb 27 02:06 remount_service.cpp
-rw-rw-r-- 1 nanik nanik  19120 Feb 27 02:06 services.cpp
-rw-rw-r-- 1 nanik nanik   4291 Feb 27 02:06 set_verity_enable_state_service.cpp
-rw-rw-r-- 1 nanik nanik  29034 Feb 27 02:06 shell_service.cpp
-rw-rw-r-- 1 nanik nanik   1694 Feb 27 02:06 shell_service_protocol.cpp
-rw-rw-r-- 1 nanik nanik   5954 Feb 27 02:06 shell_service_protocol_test.cpp
-rw-rw-r-- 1 nanik nanik  10337 Feb 27 02:06 shell_service_test.cpp
-rw-rw-r-- 1 nanik nanik  25244 Feb 27 02:06 sockets.cpp
-rw-rw-r-- 1 nanik nanik  10118 Feb 27 02:06 socket_test.cpp
-rw-rw-r-- 1 nanik nanik   8152 Feb 27 02:06 sysdeps_test.cpp
-rw-rw-r-- 1 nanik nanik   1990 Feb 27 02:06 sysdeps_unix.cpp
-rw-rw-r-- 1 nanik nanik 103087 Feb 27 02:06 sysdeps_win32.cpp
-rwxrwxr-x 1 nanik nanik   8478 Feb 27 02:06 sysdeps_win32_test.cpp
-rw-rw-r-- 1 nanik nanik   1931 Feb 27 02:06 test_track_devices.cpp
-rw-rw-r-- 1 nanik nanik  31093 Feb 27 02:06 transport.cpp
-rw-rw-r-- 1 nanik nanik  14945 Feb 27 02:06 transport_local.cpp
-rw-rw-r-- 1 nanik nanik   6907 Feb 27 02:06 transport_test.cpp
-rw-rw-r-- 1 nanik nanik   2458 Feb 27 02:06 transport_usb.cpp
-rw-rw-r-- 1 nanik nanik  18420 Feb 27 02:06 usb_linux_client.cpp
-rw-rw-r-- 1 nanik nanik  19763 Feb 27 02:06 usb_linux.cpp
-rw-rw-r-- 1 nanik nanik  18436 Feb 27 02:06 usb_osx.cpp
-rw-rw-r-- 1 nanik nanik  20215 Feb 27 02:06 usb_windows.cpp
```



```
LOCAL_CFLAGS += \
    $(ADB_COMMON_CFLAGS) \
    -D_GNU_SOURCE \
    -DADB_HOST=1 \

LOCAL_MODULE := adb
LOCAL_MODULE_TAGS := debug

LOCAL_STATIC_LIBRARIES := \
    libadb \
    libbase \
    libcrypto_static \
    libcutils \
    liblog \
    $(EXTRA_STATIC_LIBS) \

# libc++ not available on windows yet
ifneq ($(HOST_OS),windows)
    LOCAL_CXX_STL := libc++_static
endif
```

```
LOCAL_CFLAGS += -DALLOW_ADBD_NO_AUTH=$(if $(f

ifneq (,$(filter userdebug eng,$(TARGET_BUILD
LOCAL_CFLAGS += -DALLOW_ADBD_DISABLE_VERITY=1
LOCAL_CFLAGS += -DALLOW_ADBD_ROOT=1
endif

LOCAL_MODULE := adbd

LOCAL_FORCE_STATIC_EXECUTABLE := true
LOCAL_MODULE_PATH := $(TARGET_ROOT_OUT_SBIN)
LOCAL_UNSTRIPPED_PATH := $(TARGET_ROOT_OUT_SB
LOCAL_C_INCLUDES += system/extras/ext4_utils

LOCAL_STATIC_LIBRARIES := \
    libadbd \
    libbase \
    libfs_mgr \
    liblog \
    libcutils \
    libc \
    libmincrypt \
```

Figures 4 and 5 - from left to right: ADB target and ABDB target.

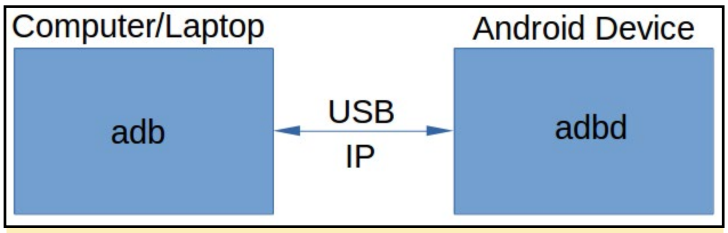


Figure 6 - ADB in host and ADBD in Android device

Both adb and adbd are in the same code base, but what separates them during the compilation process is the use of macro. This makes the project output 2 different apps, as seen from the Android.mk in Figures 4 and 5. The way adb runs inside the host is different than inside the device.

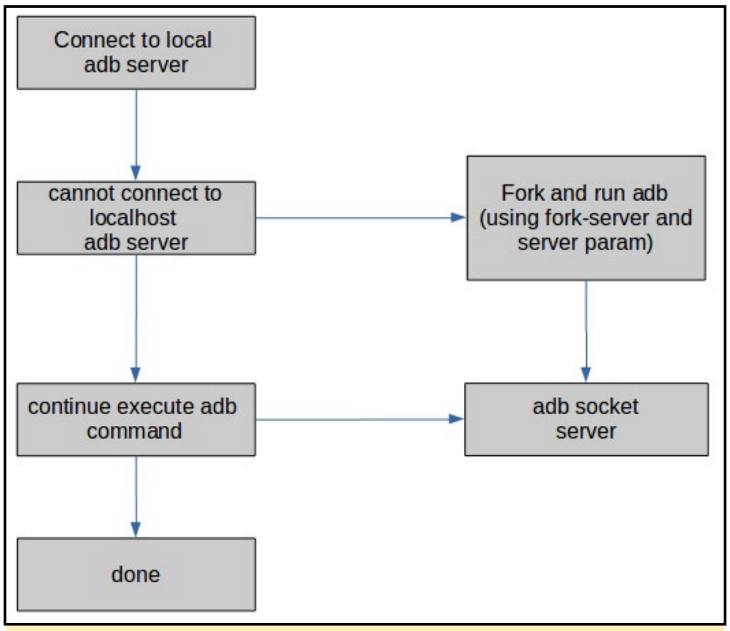

Connect to local adb server
cannot connect to localhost adb server
Fork and run adb (using fork-server and server param)
continue execute adb command
adb socket server
done

Figure 7 - Process flow running adb in host

As shown in the process flow of Figure 7, what really happens internally running in the host computer is that adb spawns off a new adb process that will be used to communicate with the remote Android device. For example, if you run the command “adb devices”, adb will have 2 different process running in the computer: one running as a server doing the communication with the adbd in the Android device, and the other process interpreting the command that the user has instructed, which is “devices” in our example. Figure 8 outlines how adb will look like in the host computer when the user executes a command such as “adb devices.”

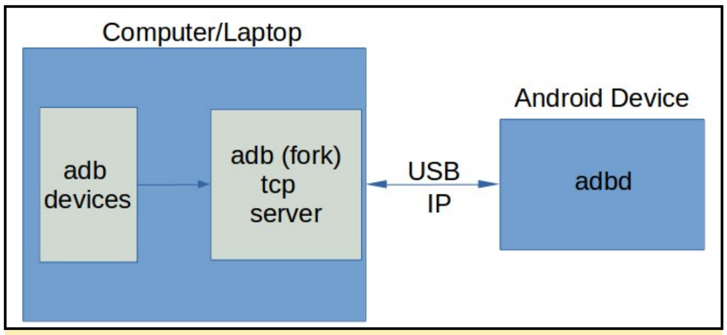


Figure 8 - 2 processes running inside host

What this means is that whenever you use adb to connect to your Android device to do an operation, after the operation completes, there is a single process that is still running in your computer connected to your Android devices. This long running process allows you to continue performing operations on your devices without losing any connection. In part 2, we will look more in depth into the different parts of adb using the source code to dig deeper and understand it.

ANDROID NAVIGATION

USING AN INFRARED REMOTE CONTROL

by Lorenzo Carrieri

In this project, we will create an IR based remote control that we can use to remotely operate Android, removing the need for a touch interface. This can be used as a simple remote control, for a home-made Android TV system based on an ODROID or, for instance, as an interface between the automotive CAN bus network and the Android operating system.

This interface has been developed and tested to be fully functional with an ODROID-C2 but, it should work just as well with an ODROID-C1/C1+, running Android Lollipop 5.1.1 v3.4.

Hardware requirements

- ODROID-C2
- 1x Arduino Nano
- 1x mini breadboard
- 4x buttons
- 4x 10 kΩ resistors
- 1x 100 Ω resistor
- 1x IR led

Software requirements

- Android Lollipop 5.1.1 v3.4 from Hardkernel
- Arduino IDE v1.8.1
- Arduino IRremote library, http://bit.ly/1Isd8Ay
- Android file manager with text file editing capabilities

Building the IR remote control

In this example, we will create an IR interface capable of opening the Android app switcher, navigating through the app list, and starting a desired app. Now, let's build the IR remote control!

According to the wiring diagram shown in Figure 1, connect digital pin 3 to the IR LED anode bringing the cathode to ground through a series of 100Ω resistors. Next, prepare a group of four buttons which connect the Arduino Nano +5V power out pin to four different digital input pins, in this case 5, 6, 7 and 8. Remember to connect a 10 kΩ pull down resistor

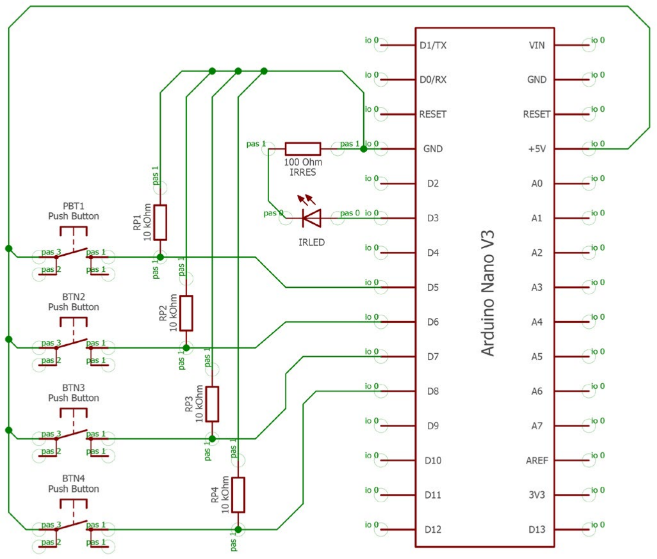
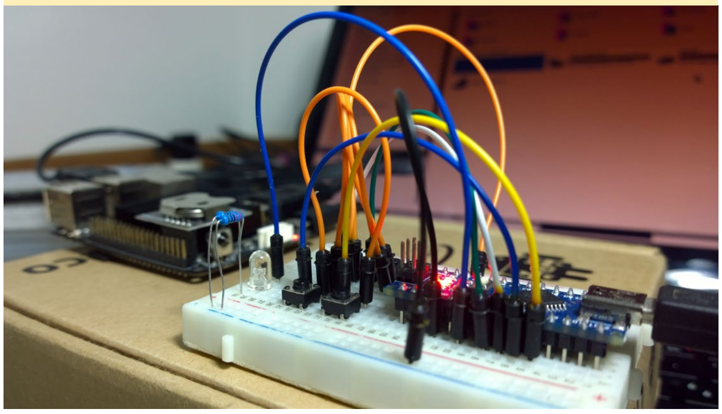


Figure 1 - Wiring diagram

Figure 2 - IR remote controller



between each input pin and the ground through the button, as shown in the wiring diagram.

The remote.conf file

After building the remote, we need to know what this remote will “say” to the ODROID-C2 IR receiver. Let’s open the file called remote.conf, located in the ‘\system\etc’ folder. In the file, you will find something similar to the following:

```
key_begin
        0x88 113
        0xdc 116
        ...
key_end

repeat_key_begin
        0x88 113
        0xdc 116
        ...
repeat_key_end
```

Look at the first line after the ‘key_begin’ statement; it is made up of two parts: a binary string (hexadecimal coded), an IR action code, followed by a keycode number. The first part is the digital word the ODROID-C2 wants to read from its IR receiver in order to execute the Android event associated with the corresponding keycode number. A full list of Android event keycodes is contained in the ‘/usr/keylayout/generic.kl’ file. For this example, we find the following keycodes:

Android Event	Description	Android Keycode
APP_SWITCH	Open the app switcher	580
DPAD_UP	Move up in a list	103
DPAD_DOWN	Move down in a list	108
DPAD_CENTER	Confirm selection	97

Table 1 - Android event keycodes

Note that, except for APP_SWITCH, the other three events are already present in remote.conf file, but they need a little modification. For the moment, just add a new line for the APP_SWITCH event in each section of the remote.conf file enclosed by the _begin and _end statements:

```
key_begin
        0x88 113
        0xdc 116
        ...
        0x12 580     (this is the new line added)
key_end
```

```
repeat_key_begin
        0x88 113
        0xdc 116
        ...
        0x12 580     (this is the new line added)
repeat_key_end
```

For the next step, ensure that IR debug mode is on. If it is not, you can activate the debug mode through the remote.conf file by changing debug_enable option from 0 to 1:

```
debug_enable = 1
```

If altered, save and reboot ODROID-C2 to apply the change.

Arduino code for the IR remote control

First, install all the needed libraries as described in the Installation section at http://bit.ly/1Isd8Ay. Then, open a new sketch and paste the following code:

```
/*
* ODROID IR SEND TEST
* An IR LED must be connected to Arduino PWM pin 3.
* Version 0.1 March, 2017
*/

#include <IRremote.h>                // we include
the IR remote library
IRsend irsend;

void setup()
{
pinMode(5, INPUT);           // now we set all the
input pin
pinMode(6, INPUT);           // to which we will as-
sociate
pinMode(7, INPUT);           // the selected Android
events
pinMode(8, INPUT);
}

void loop() {
// irsend.sendNEC(IR code, 32) is the function that
generate and send the 32 bit IR binary word
// in compliance with the IR NEC protocol (the same
of the ODROID IR receiver)

if (digitalRead(5)==HIGH) {  // open the App switcher
irsend.sendNEC(0x4db2bb00,32);
}

else if (digitalRead(6)==HIGH)
{ // simulate tap on the screen
(press OK)
irsend.sendNEC(0x4dce00,32);
}

else if (digitalRead(7)==HIGH) {
// simulate scroll down
irsend.sendNEC(0x4db2d200,32);
}

else if (digitalRead(8)==HIGH) {
// simulate scroll up
irsend.sendNEC(0x4db2ca00,32);
}
}
```

As you can see in the code, the function irsend.sendNEC(IR code,32) takes an IR code as a parameter which is a 32bit long integer, written here as hex, that contains the IR emitter identifier and the IR event code. In this code we see it formatted in hex such as this:

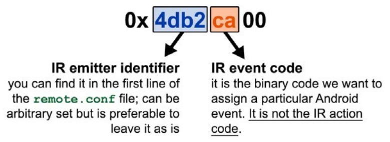

0x 4db2 ca 00
IR emitter identifier
you can find it in the first line of the remote.conf file; can be arbitrary set but is preferable to leave it as is
IR event code
it is the binary code we want to assign a particular Android event. It is not the IR action code.

Figure 3 - Structure of the IR code

0x4db2ca00, as shown in Figure 3.

For this example, we have four command buttons and four different 'if' statements that, on true, call the function irsend to generate the corresponding IR signal, compliant with the NEC protocol, on digital pin 3. Now we can load the sketch on to the Arduino Nano.

Configuring the remote.conf file

The final step of this example is the configuration of the remote.conf file, which links the attribution of the correct IR action code to the desired Android event. Note that, at this time, we have assigned a specific IR event code to each of the Android events, but we don't know the corresponding IR action code to write in the remote.conf file. So, connect your ODROID-C2 board to the PC with developer option enabled and open an ADB session in Windows command prompt and follow this procedure:

- press and hold one of the buttons on the IR controller, for instance the one associated with the app switcher
- without releasing the remote control button, enter the command dmesg into the ADB prompt
- release the button.

Looking at the dmesg result, the last rows should show something like this:

```
remote: scancode is
0x004b,invalid key is 0x0000.
```

The scancode 0x004b is the IR code read by ODROID-C2 in which 4b is the IR action code that we have been searching for. Open the remote.conf file, go to the line we previously added related to the APP_SWITCH event, and write the correct IR action code. Following the above example, we would now have the following:

```
key_begin
        0x88 113
        0xdc 116
        ...
        0x4b 580     (this is the
new line added with the correct
IR action code)
key_end

repeat_key_begin
        0x88 113
        0xdc 116
        ...
        0x4b 580     (this is the
new line added with the correct
IR action code)
repeat_key_end
```

Repeat this procedure for all the buttons, then save and reboot the ODROID-C2. If everything went correctly, you should now have a fully working home-made ODROID IR remote control.

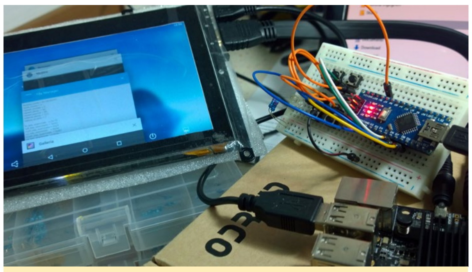


Figure 4 - Complete system

Notes

The procedure described in this guide lets the user start a specific Android event using an external IR remote control. Since Android Lollipop 5.1.1 v2.9, it has been possible to start a selection of four different apps directly through GPIO pins. This service recalls the Android events F7, F8, F9, and F10 which can be associate to a specific app simply using the ODROID Utility application. This result can be achieved also using an external IR remote control. Simply associate the keycodes 65, 66, 67, and 68 to the events for F7, F8, F9, and F10 respectively, using the IR action code to map the keycodes to events as demonstrated in this article.

Hopefully a remote control for a remote control will be the subject of Lorenzo's next article

YEAH, IF YOU COULD MAKE A REMOTE CONTROL THAT BRINGS ME THE REMOTE CONTROL
THAT'D BE GREAT

LINUX GAMING

F-ZERO SERIES CAR RACING

by Tobias Schaaf

The F-Zero series, which came out for a lot of different systems, most of which had limited capabilities, turned out to be known for its very fast past racing action. It came out for the SNES first, and introduces what is now called Mode 7 scrolling, which combines scaling and positioning/turning of layers to make an effect that looked like it was in 3D. The same technique was later used for Super Mario Kart. You choose one of four racers and get right into the action.

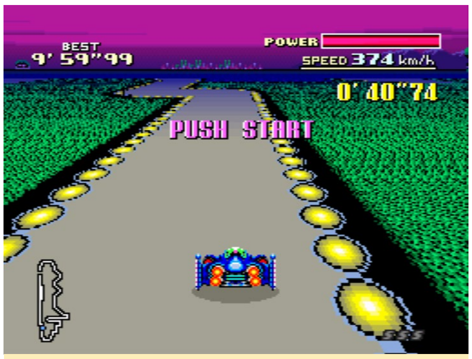


Figure 1 - F-Zero on the SNES in Mode 7

The game is very fast compared to other racing games such as Super Mario Kart and is, at times, very unforgiving. For example, if you hit the edge of the road, you will be damaged and lose speed. You can also bounce off the wall and other racers, which can lead to some flipper effects and cause you to lose speed and power/health very quickly. The track also has some obstacles like dirt patches that slow you down, or pads that make you jump or give you a boost, not all of which work in your favor. At the start of each track, there is a recharging field that can restore your power if you received damage.

With each round you complete, you fill one boost, signaled by a green "S" in the lower right corner of the screen. You can stack up a maximum of 3 boosts, which allow you to catch up quickly if you were slowed down by hitting another car or a wall.

Aside from the racers that you compete against, the tracks will have random cars popping up once you hit the second round. These cars can slow you down, or you get pushed against a wall. There's even a blinking car showing up at the track occasionally, which you should avoid hitting at all costs, since this one is going to explode, dealing quite a bit damage and completely throwing you off course. Some courses even have magnetic strips or wind that lets your racer slowly drift to the side.

The tracks are fairly tough, and I enjoyed playing it a lot, although I'm normally not a fan of racing games. The old Super Mario Kart really does nothing for me, and feels very slow compared to F-Zero. Unlike Mario Kart, the music and game play in F-Zero fit perfectly together.

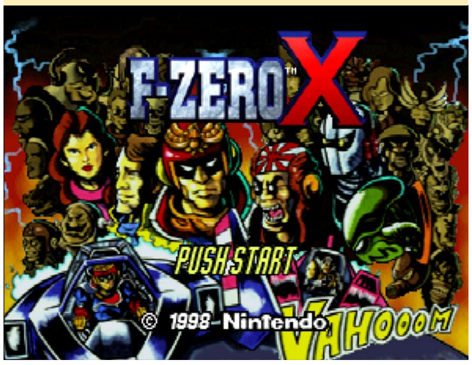
F-Zero had 5 more releases for the SNES as well as the Super Famicom System, which were only available in Japan. It used the Satellaview satellite modem add-on of the Super Famicom, which allowed you to download a game to memory and play it until you replaced it with another game.

The first game was released in 4 episodes: F-Zero Grand Prix - Knight League, Queen League, King League, and Ace League. Unfortunately, I couldn't get any of these to work, but I read that there is a patch that should work with BSNES to get the games to work again. F-Zero Grand Prix 2 is working fine, although it's just the Ace League that you can play. It feels even faster than the original F-Zero game with new exciting tracks, and is a lot harder to play.

F-Zero X

The next release in the series was a jump into 3D on the Nintendo 64,

Figure 2 - F-Zero X on the N64



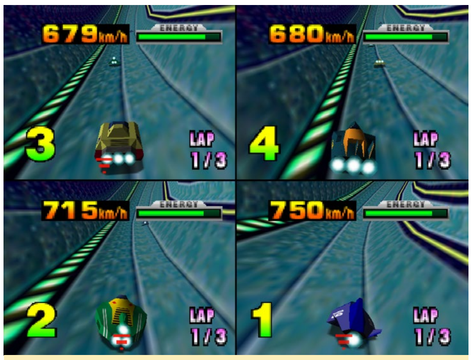


Figure 3 - F-Zero X supports up to 4 player on split screen

where the series was picked up and improved a lot, which is probably the best known game of the series.

Right at the start, you're greeted with awesome fast-paced music, and everything in this game screams "speed!". The game was intentionally produced with a lower graphics quality, which means the line of sight is somewhat limited. There are only a few objects apart from the racers and the track, and there's rarely any sky or other background elements that can be seen. This was done to keep the frame rate always at 60 FPS, which makes the game feels very fast and fluent. It all worked perfectly to create an awesome racer game.

As mentioned earlier, the game had many changes as compared to the SNES version, such as racing against 29 other racers rather than just 3. The tracks you could race got a 360° upgrade, which was fully used on the N64. You could race in tubes that allowed you to drive on the ceiling, or just spin around in circles. You could even race on the outside of a tube, which means that instead of being limited by walls in all directions, you could only see the sky, which made it seem like you could fall off of the track at any time. In fact, falling off of the track was a distinct possibility, and some tracks, with their half or quarter pipes, were actually designed to throw you off the track to crash and burn. The game came with bend curves, loops, and different kinds of jumps.

The way boosts could be used was updated as well. Starting from the second round, you can use boosts now, but not like in the SNES version which limited you to one boost per round; you can use them as long as you have Energy. Energy is similar to the power bar on the SNES version, and reduces if you hit obstacles like other racers. Starting with the second round, you can use the same energy to activate boosts. Each time you activate it, you'll lose some energy, and with that also some of your "health", so you need to be careful how often you use your boost.

Aside from the 4 racers of the original game, there are additional 26 racers that you can unlock and choose from, which gives the game a much bigger variety. The developers also added new game modes: Time Attack, Death Race and even a multiplayer spit-screen mode called VS Battle. You could unlock not only new racers, but also new cups, with the final cup being the X-Cup with auto-generated maps, which means that the tracks were different each time you raced them.

The game runs very nicely if you run it on an ODROID-XU3/XU4 or an Exynos 4-based ODROID (X, X2, U2/U3) and is really fun to play. The emulator supports rumbling of the joystick, which gives you a little extra depth when you hit a wall or an enemy or just crash and burn.

Figure 4 - In F-Zero X you race against 29 other racers

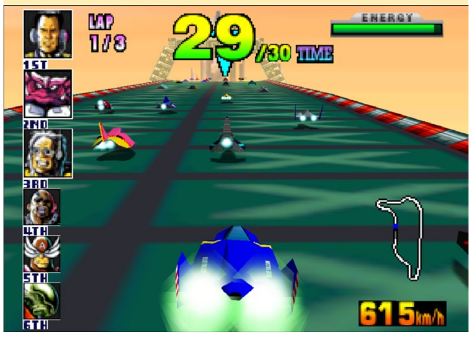


Figure 5 - Spinning in circles in F-Zero X

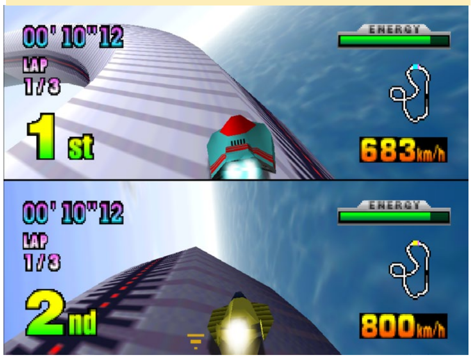


Figure 6 - Recharging your energy meter in F-Zero X

F-Zero X Expansion Kit

Two years after the release of F-Zero X, Nintendo developed an expansion kit for the game. This was only available in Japan and only for the 64DD (the Nintendo 64 Disk Drive). It was an expansion disk that added new racers, new cups and courses, new music, and two very special extra features.

This expansion kit allowed you to design your own racer for the game where you could put the racer together very much to your liking, paint it the way you wanted, added logos and other enhancements. That alone was already a very cool feature, but the developers went further and included a course editor. This allowed you to create your own courses to race on no matter how crazy they were, with twists, curves, and spins. The expansion was praised for this feature the most. Unfortunately, you currently can't use it on the ODROID, so I couldn't test it myself.

F-Zero on GBA

The F-Zero series moved then to the Game Boy Advanced (GBA) on which it had three releases. In 2001, one year after the 64DD expansion, the first game called F-Zero – Maximum Velocity was released, and was actually a launch title for the GBA. This time the F-Zero series went back to its origin and the game pretty much looked and played like the old SNES version. Thanks to the higher processing power of the GBA compared to the SNES as well as the higher number of colors, the game runs even faster and smoother on the GBA than the SNES version does, although it has a lower resolution, due to the smaller screen of the GBA. I went back to play the SNES version for a little while after I played the GBA version, and it really felt like suddenly the game was much slower. While the SNES version has a higher resolution and cars look much better, the background in the SNES version was rather dull compared to the GBA version. I prefer the GBA version over the SNES version even though the resolution is somewhat lower, because the game is faster and overall more fun.

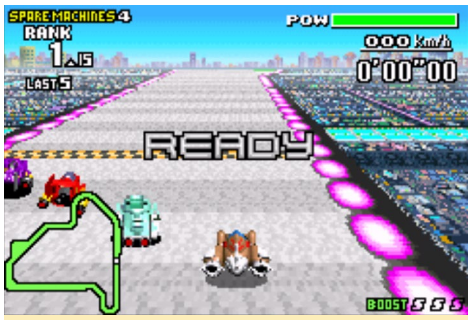


Figure 7 - F-Zero Maximum Velocity on the GBA, looks much like the SNES version

In 2003, F-Zero GP Legend was released for the GBA. This game is closer to the N64 version, and also based on an anime with the same name. In GP Legends, similar to the N64 version, you now race as one of 30 racers in a Grand Prix, although it still feels closer to the SNES version because you only see a few racers on the track. However, you can play and unlock up to 34 racers/characters, just like the N64 version. Other features of the N64 version were also introduced into the game, such as the power meter doubling both as health and booster energy. It added a couple new game modes such as time attack, and a story mode that allowed you to follow the story of 8 out of the 34 racers/characters. The story is told through nice anime-style cut-scenes.

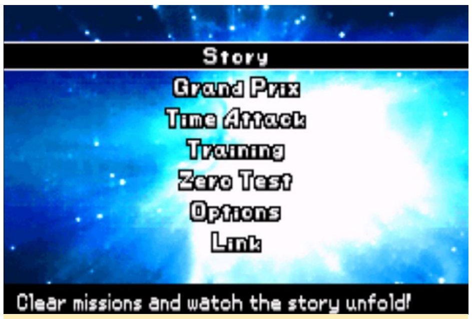


Figure 8 - New Game modes in F-Zero GP Legends

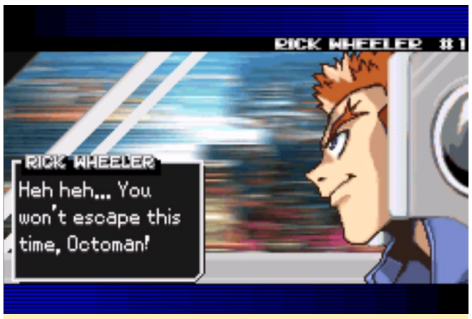
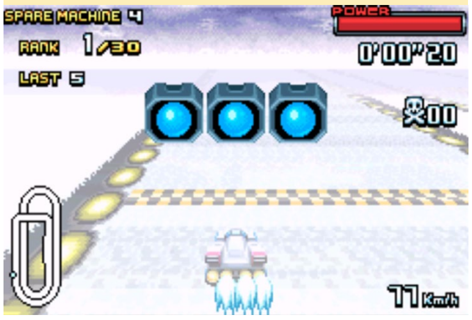


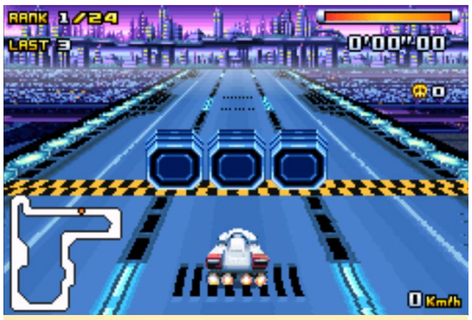
Figure 9 - Story mode in F-Zero GP Legend is told in Anime cut-sequences

The game updated the graphics a little as well, and shows off some weather effects. However, the game has some issues with bouncing off of walls and other racer, which leads to terrible pinball behavior, where you have no control over your racer and just get slammed into the walls left and right and can do nothing against it. That's very frustrating and really interferes with the gaming experience. The symbiosis of SNES and N64 elements is actually quite nice, although the graphics are quite a step back compared to the N64 version.

The final installment of the series was released in 2004 and is called F-Zero Climax, which was only released in Japan for the GBA. F-Zero Climax is actually the best of the 2D games in my opinion. It has everything that made the SNES version great, and adds the new features from F-Zero GP Legend, without the ridiculous pinball effects. This one is really fun to play, and you have lots of things to unlock while you complete races. It even comes with a track editor that allows you to create your own crazy tracks.

Figure 10 - Weather effects in F-Zero GP Legend, fog in misty places





Figures 11 and 12 - F-Zero Climax is probably the best you can get on a system like the GBA with nicely drawn backgrounds and smooth and fast gameplay

Other F-Zero titles

In 2003, there were also two new 3D games released in the series for the Nintendo GameCube and arcade: F-Zero GX and F-Zero AX. Both of these games were similar and actually could share game data, means you could take your character from the Game Cube and put it into the arcade machine to unlock extra content. This was very unique, and the Arcade system actually looks really awesome with a cockpit and pedals and nice steering wheel: https://www.youtube.com/watch?v=MAFZvKkVt10.

The graphics are really nice and the gameplay is ultra fast which makes this game really fun. Unfortunately, neither the GameCube version nor the arcade version can currently run on ODROIDs, but this might change with future boards. However, if you have a PC with Dolphin Emulator for GameCube, you can play the game in up to 4k resolution and turn on other effects depending on what your PC supports, making this game graphically very impressive. You should definitely give it a try if you like racing games.

Notes

I'm very new to the F-Zero series, but I enjoyed it a lot. I liked the SNES version when I first played it for being so much faster than Super Mario Kart, but after playing the GBA versions, even the SNES version seems slow. I definitely prefer F-Zero Climax over GP Legend, although the story mode is a nice addition. F-Zero X on the N64 is great as well, especially if you want to play with friends. The 3D spinning is very fun, and I really enjoy the music of the N64 version. This game just make you feel the speed, and the rumbling support gives it a little more of an arcade feel. I hope that one day we will be able to play the GameCube version on a ODROID as well, and have all the F-Zero glory we want right there in one device.

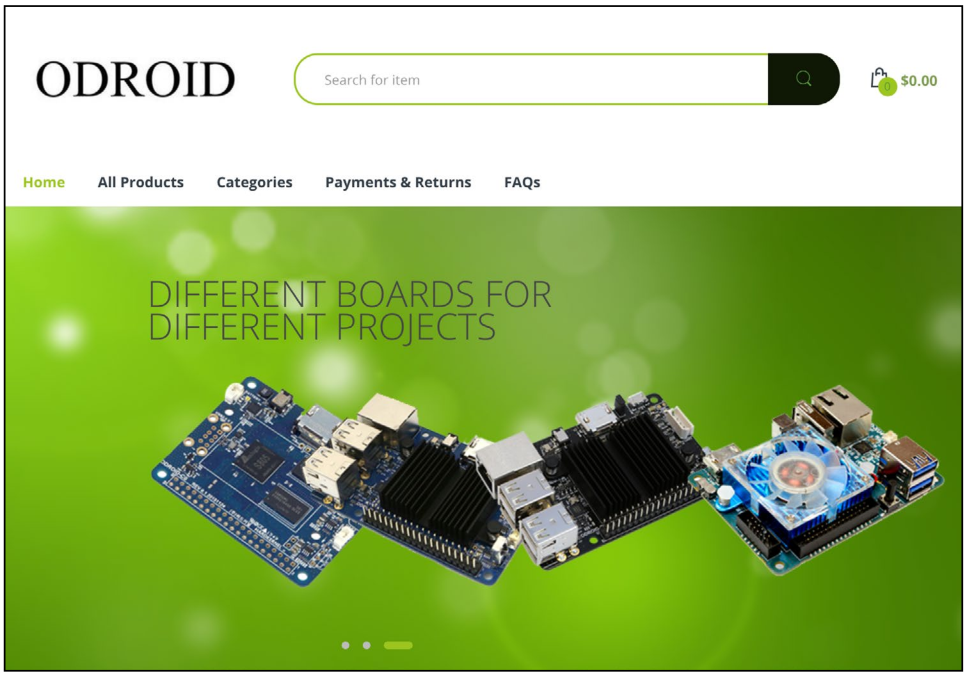
Bruno and Rob Roy really take their F-Zero games seriously

A NEW ODROID STORE IS OPEN IN THE US

VISIT ODROIDINC.COM

by Rob Roy (robroy)

A new Hardkernel distributor has just opened an online store in the United States, which offers their complete catalog of products, including the ODROID-XU4, ODROID-C0, ODROID-C1+ and ODROID-C2. The site calculates bulk discounts for a number of products and several shipping options for fast delivery. It will be opening in mid-April, so make sure to visit the new website at `www.odroidinc.com` for all of your single board computer needs.



RETROPIE VERSION 4.2

NOW OFFERING NATIVE ODROID-C2 SUPPORT

edited by Rob Roy

RetroPie allows you to turn your Raspberry Pi or PC into a retro-gaming machine. It builds upon Raspbian, EmulationStation, RetroArch and many other projects to enable you to play your favorite Arcade, home-console, and classic PC games with minimum set-up. For power users, it also provides a large variety of configuration tools to customize the system as you want. RetroPie sits on top of a full OS, you can install it on an existing Ubuntu image, or start with the RetroPie SD image and add additional software later.

Emulators

An emulator is software that makes a computer behave like another computer, or in the case of RetroPie, a computer that behaves like a video game console such as the Super Nintendo. The RetroPie SD image comes pre-installed with many different emulators. ROMs are digital versions of game cartridges. Loading up a ROM in an emulator is the equivalent of putting a cartridge in a game console. ROMs are copyrighted content and as such are not included with RetroPie.

Version 4.2

A lot has happened since 4.1, with updates to EmulationStation adding video support and fixing the dreaded white screen of death. Many packages have been updated, and RetroPie 4.2 includes the latest RetroArch v1.5.0 as well as Kodi 17 (installed optionally). RetroPie 4.2 also includes initial support for the ODROID-C2 board, which is installed on top of the ODROID-C2 Ubuntu minimal image. You'll also notice that the documentation has received a much needed update at http://bit.ly/2pdVK42. There are many other changes, including usability improvements, and bug fixes – for more details please see the changelog below.

You can download a 4.2 image from http://bit.ly/1WB25BO. For new installations, please follow the installation instructions at http://bit.ly/2pdZNO2. If updating from 4.0.x, you should make a backup first, then choose "Update all installed packages" from RetroPie-Setup main menu. Anyone upgrading from 3.x will need to update the RetroPie-Setup script first, as detailed at http://bit.ly/2pXcV80.

You can also install RetroPie on top of an existing Raspbian setup, or on top of Ubuntu on a PC/Odroid-C1/C2. Links to the relevant instructions can be found in the downloads area. Thanks to all those who contributed to this release with a special mention to @fieldofcows for his excellent EmulationStation improvements.

Changelog

- EmulationStation Improvement: Video Support, White Screen of Death Fix
- Support for the ODROID-C2 on top of the Ubuntu 16.04 minimal image
- Kodi 17 now installable from optional packages
- AdvanceMame has been updated and split into three separate packages: 0.94, 1.4 and v3.3
- Updated to RetroArch v1.5.0
- To match upstream changes, lr-mupen64plus has been renamed to lr-parallel-n64, and lr-glupen64 has been renamed to lr-mupen64plus
- Fixed launching Pixel desktop and other X11 apps from EmulationStation
- Fixed problems building Zdoom, ResidualVM and Mupen64Plus and PPSSPP
- Doom ports will automatically add launch scripts if it finds doom1.wad, doom2.wad, tnt.wad, or plutonia.wad
- lr-snes9x emulator added, which is a libretro port of the current snes9x codebase
- Added Amiberry (an Amiga emulator), which is an updated fork of uae4arm, with more features
- Multi disk zip support for Vice (C64 emulator), fs-uae, uae4arm and Amiberry (Amiga)
- You can now launch Amiga disk images directly from EmulationStation with uae4arm and Amiberry
- Standalone version of Stella (At-

ari 2600 emulator) updated to v4.7.3
- usbromservice, which supports mounting of USB sticks over ~/RetroPie to keep ROMs on USB
- Ability to set custom ES themes in configs/all/platforms.cfg, which can override any setting in RetroPie-Setup/platforms.cfg
- SDL2 updated to 2.0.5
- @Sselph's scraper updated to the latest version, and new options added. Scraper has been moved to optional packages and needs to be installed before it will show up in configuration / tools
- Include PowerBlock and ControlBlock driver packages
- Input configuration script for Daphne
- RetroPie setup menus now works with all connected joysticks, although mapping is still hardcoded
- Updated RPI detection code to support BRANCH=next firmware/kernel
- Overhaul of the runcommand launch script
- Raspbian Wheezy support removed
- New packages added to experimental section

Building RetroPie

To build RetroPie for the ODROID-C2, start with a pre-built image of Ubuntu from Hardkernel's website at http://bit.ly/1dFLsQQ (http://bit.ly/1OU4kbl). Extract the .xz file with a program like 7zip, then write the .img file to your SD card or EMMC module with Win32DiskImager (http://bit.ly/2pe19YV). Unlike the RetroPie SD Image, the ODROID image will auto-expand the filesystem, so there is no need for that step here. To start, type the following commands into a Terminal window:

```
$ sudo apt-get update && sudo apt-get upgrade
$ sudo apt-get install -y git
$ cd
$ git clone --depth=1 https://github.com/RetroPie/
RetroPie-Setup.git
$ cd RetroPie-Setup
$ sudo ./retropie_setup.sh
```

If you have issues while compiling modules and it freezes up on you, then you need to tell it to only compile with one core by running the setup script:

```
$ sudo MAKEFLAGS="-j1" ./retropie_setup.sh
```

All modules can be installed from the RetroPie Setup Script. The two main packages you need in order for the majority of your system to run are RetroArch and EmulationStation. You can then choose your emulators from the available options.

Fixing non-working sound

If you have troubles with no sound or sound stuttering badly in menu or game, check your CPU usage via top or htop. If pulseaudio is using more than 20%, then it may be the culprit. In my case, it was using 80%! I had only one sound card output (HDMI) so I completely removed pulseaudio. For Ubuntu 14, this is done with a single command:

```
$ sudo apt-get --purge remove pulseaudio
```

To disable pulseaudio instead, type the following commands into a Terminal window:

```
$ mkdir ~/.pulse
$ echo "autospawn=no" >> ~/.pulse/client.conf
$ pulseaudio -k
```

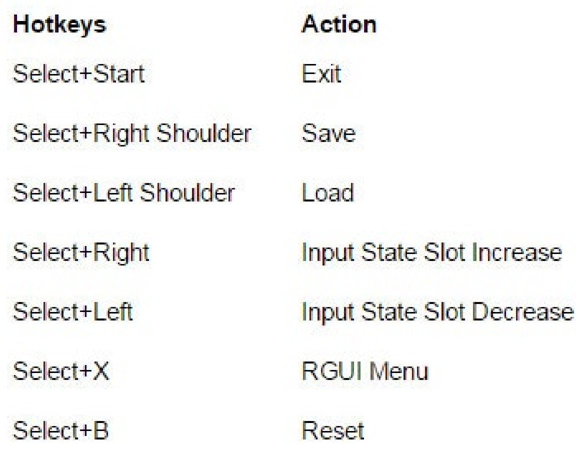
Configure controllers

On first boot, your filesystem will be expanded automatically, and you will then be welcomed with the controller configuration menu for both Emulationstation and RetroArch Emulators. Hold down any button on your keyboard or gamepad and the name will appear at the bottom and then open up into a configuration menu. Follow the onscreen instructions to configure your gamepad. If you run out of buttons just hold down a button to skip each unused button. When asked to press "OK", press the button you have configured as "A".

If you wish to configure more than one controller, you can do so from the start menu of emulationstation. For more details on manual controller configurations, please refer to http://bit.ly/2ooycuf.

Hotkeys

Hotkeys enable you to press a combination of buttons to access functions such as saving, loading, and exiting emulators. The following chart shows the default hotkey combinations. By default, the hotkey is selected, so that means you hold down select while pressing another button to execute a command. Hotkeys are only specific to the retroarch/libretro based emulators.



Hotkeys	Action
Select+Start	Exit
Select+Right Shoulder	Save
Select+Left Shoulder	Load
Select+Right	Input State Slot Increase
Select+Left	Input State Slot Decrease
Select+X	RGUI Menu
Select+B	Reset

Installing additional emulators

On RetroPie 4.0+, not everything is installed by default. The pre-made images contain the best working emulators for each system supported by the hardware. This should cover everything most users would be doing. Ports like quake and doom and some other emulators like ScummVM can be installed later.

Software can be installed from the RetroPie-Setup script, which is accessible from the RetroPie menu on EmulationStation. Once there, you can navigate to "Manage Packages", where you will see various sections. In each section are lists of packages that can be installed along with what is currently installed. Stable additional packages are under the "Optional" section, with more unstable packages listed under experimental. The packages are ordered first by type, then alphabetically. By selecting a package you can choose to install it or remove it. Some packages also have additional configurations.

Transferring ROMs

Due to the nature/complexity of Copyright and Intellectual Property Rights Law, which differs significantly from Country to Country, ROMs cannot be provided with RetroPie and must be provided by the user. You should only have ROMs of games that you own. There are three main methods of transferring ROMs: USB, SFTP, and Samba.

To transfer via USB, Ensure that your USB is formatted to FAT32 or NTFS, then create a folder called retropie on your USB stick. Plug it into the ODROID and wait for it to finish blinking, then pull the USB out and plug it into the computer containing the ROMs. Add the ROMs to their respective folders in the retropie/ROMs folder, then plug it back into the ODROID and wait for it to finish blinking. Refresh EmulationStation by choosing "Restart EmulationStation" from the Start menu.

To transfer via SFTP, you first need to enable SSH (http://bit.ly/2oDSP0S). The default username is "pi", and the default password is "raspberry". You can also login as root if you wish to change more files than just the ROMs, but you first need to enable the root password, which is explained at http://bit.ly/2pdX3QG.

To transfer via Samba from a Windows machine, type \\retropie into the Windows Explorer, or type \\<ip-address>. If using OS X, select "Go" in the Finder, and "Connect to Server". Type smb://retropie (or use the IP address) and select "Connect".

Playing games

After you've added your ROMs, you need to restart EmulationStation in order for them to show up by using the Start menu, or by rebooting your ODROID using the "sudo reboot" command. Please refer to the Retropie documentation at http://bit.ly/2pdVK42 for more detailed information on individual emulators and advanced settings. For topics not covered in the documentation, you can refer to the helpful RetroPie community forums at http://bit.ly/2oE5D7l.

The RetroPie Project is primarily maintained by a few developers who develop it in their free time. If you have found the RetroPie project useful please consider donating to the project at http://bit.ly/2q7bbs0. As you become more familiar with RetroPie, pay it forward by helping others on the forum. The RetroPie Project is what it is today because of the many contributions of the community. For comments, questions, and suggestions, please refer to the original article at http://bit.ly/2pX8Vof.

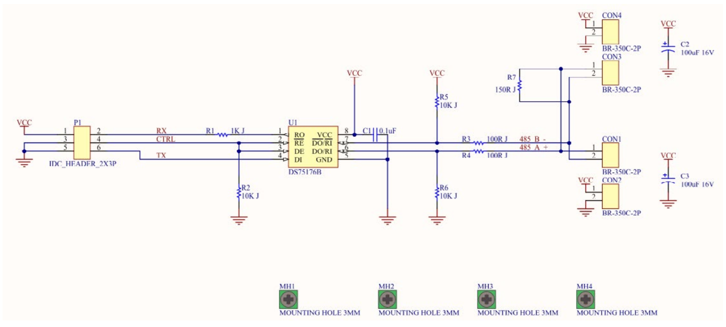
EXPLORING RS485 COMMUNICATION ON C1+ AND C2 BOARDS

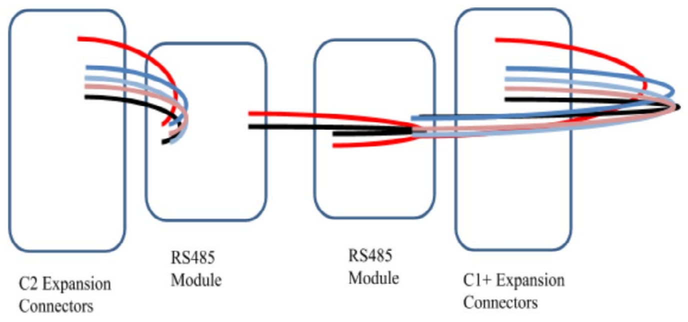
by Charles Park and Neal Kim

This article is intended to discuss the RS-485 communication protocol based on my experiences, rather than explaining the principles of RS-485 communication. I have often set up communications between systems using RS-232, which is simpler, cheaper and more widely used than other communication protocol. It involves attaching the RS-232 driver chip in any MCU (usually using an 8-bit MCU made by PIC or Atmega) and setting the UART port to talk with the other MCUs. For peer to peer communication and a short distance, I use RS-232. However, lots of industrial systems use RS-485 communication, which supports stable communication and can propagate messages to many devices from a single source at a distance.

An engineer who has experience in implementing UART drivers will find it easy to create an RS-232 or RS-485 connection, because both RS-232 and RS-485 simply specify the electrical characteristics of the generator and receiver. It does not specify or recommend any communications protocol, only the physical layer.

However, if you have a problem when building the system using RS-485, you might have to use debugging. I just want to introduce simple RS-485 communication example that is half duplex operation using 2-wire. First, wire up the devices as shown in Figures 1a and 1b. Your setup should look like the picture in Figure 2.





Figures 1a and 1b - Schematic diagram of the RS-485 module

ODROID-C2	RS485	RS485	ODROID-C1+
TXD1(PIN8)	B-	B-	TXD1(PIN8)
RXD1(PIN10)	A+	A+	RXD1(PIN10)
GPIOX.10(PIN12)	CTRL	CTRL	GPIO.Y(PIN12)
P3.3V(PIN1)	P3.3V	P3.3V	P3.3V(PIN1)
GND	GND	GND	GND

Table 1 - Pin layout chart

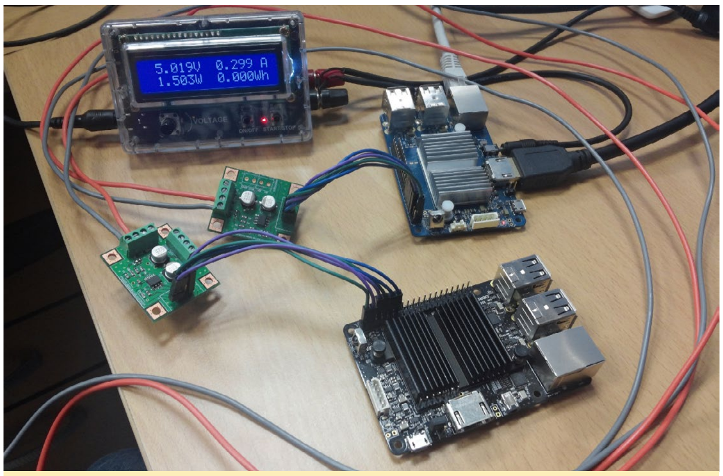


Figure 2 - ODROID devices connected via RS-485

Software

We tested the following steps using an ODROID-C2 (master) and C1+ (slave) running the latest Ubuntu images, both downloaded from http://bit.ly/1R6DOgZ. Before starting, run the following commands in a Terminal window on both devices, then reboot:

```
$ sudo apt-get update && sudo
apt-get upgrade \
  && sudo apt-get dist-upgrade
$ sudo apt-get install git vim
```

To build the test application, we need the library wiringPi. First, download the rs485_test.c source code from https://pastebin.com/GN2LxwZV. Then, download wiringPi and compile the test script into an executable with the following commands:

```
$ git clone https://github.com/
hardkernel/wiringPi
$ ./wiringPi/build
$ gpio -v
gpio version: 2.33
Copyright © 2012-2014 Gordon Hen-
derson
This is free software with ABSO-
LUTELY NO WARRANTY.
For details type: gpio -warranty
Hardkernel ODROID Details:
Type: ODROID-C1/C1+, Revision: 1,
Memory: 1024MB, Maker: Hardkernel
$ gcc -o rs485_test rs485_test.c
-lwiringPi -lpthread
```

Communication

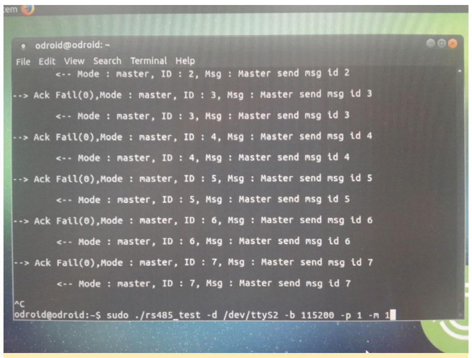
The master device transfers a message, including the ID, every second to the slave devices, then waits to get an acknowledgement (ACK) response from the slave. In case the master doesn't receive the ACK response from the slave, the master will increase the variable ack_fail_count and retransmit the message to the slave. The slave continually waits to get a message from the master, and sends an ACK response to the master immediately upon receiving the message. This

STX (1 byte)	MSG ID (4 bytes)	MSG (32 bytes)	Check data (1 byte)	ETX (1 byte)
0xFF	Message transfer ID	Messages	XOR MSG ID and MSG in packet	0xFE

Table 2 - Protocol Structure

command on the ODROID-C2 master device, which should show similar results to Figure 3:

```
$ sudo ./rs485_test -d /dev/ttyS1
-b 115200 -p 1 -m 1
```

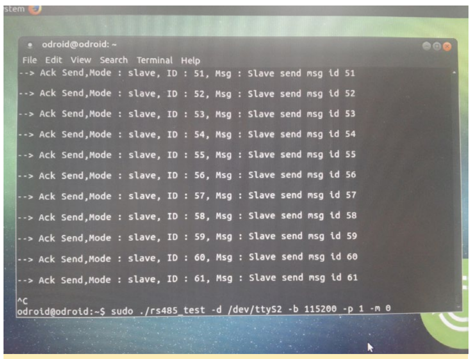


```
        <-- Mode : master, ID : 2, Msg : Master send msg id 2
--> Ack Fail(0),Mode : master, ID : 3, Msg : Master send msg id 3
        <-- Mode : master, ID : 3, Msg : Master send msg id 3
--> Ack Fail(0),Mode : master, ID : 4, Msg : Master send msg id 4
        <-- Mode : master, ID : 4, Msg : Master send msg id 4
--> Ack Fail(0),Mode : master, ID : 5, Msg : Master send msg id 5
        <-- Mode : master, ID : 5, Msg : Master send msg id 5
--> Ack Fail(0),Mode : master, ID : 6, Msg : Master send msg id 6
        <-- Mode : master, ID : 6, Msg : Master send msg id 6
--> Ack Fail(0),Mode : master, ID : 7, Msg : Master send msg id 7
        <-- Mode : master, ID : 7, Msg : Master send msg id 7
^C
odroid@odroid:~$ sudo ./rs485_test -d /dev/ttyS2 -b 115200 -p 1 -m 1
```

Figure 3 - results of RS485 test script on ODROID-C2 Master

To verify that the messages are being acknowledged, type the following command on the ODROID-C1 slave device, which should show similar results to Figure 4:

```
$ sudo ./rs485_test -d /dev/ttyS2
-b 115200 -p 1 -m 0
```



```
--> Ack Send,Mode : slave, ID : 51, Msg : Slave send msg id 51
--> Ack Send,Mode : slave, ID : 52, Msg : Slave send msg id 52
--> Ack Send,Mode : slave, ID : 53, Msg : Slave send msg id 53
--> Ack Send,Mode : slave, ID : 54, Msg : Slave send msg id 54
--> Ack Send,Mode : slave, ID : 55, Msg : Slave send msg id 55
--> Ack Send,Mode : slave, ID : 56, Msg : Slave send msg id 56
--> Ack Send,Mode : slave, ID : 57, Msg : Slave send msg id 57
--> Ack Send,Mode : slave, ID : 58, Msg : Slave send msg id 58
--> Ack Send,Mode : slave, ID : 59, Msg : Slave send msg id 59
--> Ack Send,Mode : slave, ID : 60, Msg : Slave send msg id 60
--> Ack Send,Mode : slave, ID : 61, Msg : Slave send msg id 61
^C
odroid@odroid:~$ sudo ./rs485_test -d /dev/ttyS2 -b 115200 -p 1 -m 0
```

Figure 4 - results of RS485 test script on ODROID-C1 slave

SPACETEAM

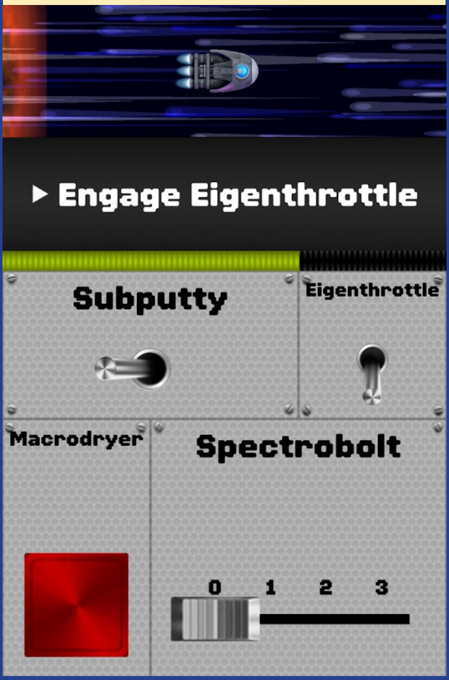
NOW YOUR FRIENDS HAVE A GOOD REASON TO YELL LOUDLY AT EACH OTHER

by Rob Roy

Spaceteam is a cooperative party game for 2 to 8 players who shout technobabble at each other until their ship explodes. You'll be assigned a random control panel with buttons, switches, sliders, and dials. You need to follow time-sensitive instructions. However, the instructions are being sent to your teammates, so you have to coordinate before the time runs out. Also, the ship is falling apart, and you're trying to outrun an exploding star. Good luck, and remember to work together... as a Spaceteam!

https://play.google.com/store/apps/details?id=com.sleeping-beastgames.spaceteam

In Spaceteam, you work together in real life to control the virtual spaceship



MEET AN ODROIDIAN

STEPHEN NEAL (@NOGGIN)

I work in the broadcast industry in a creative role, but started my career as a Research and Development engineer at a Broadcast Equipment manufacturer. I currently live in London, England, and have an engineering degree from Cambridge University.

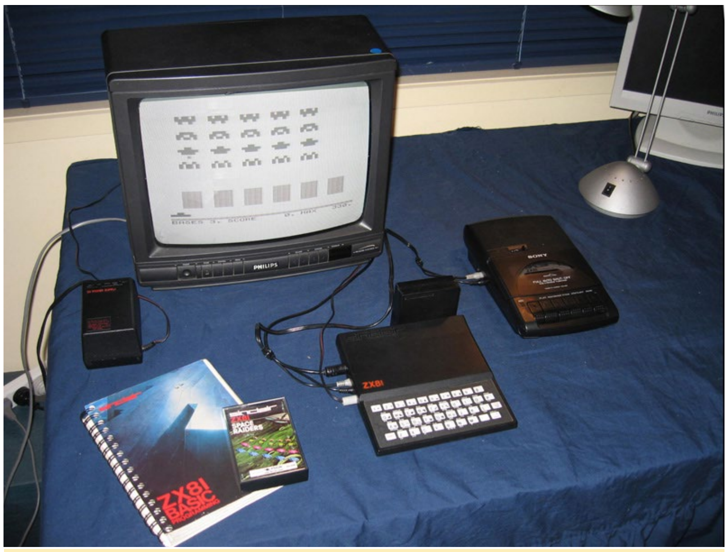
My first computer was a ZX81 from a kit in 1981 with my dad. After that, I had an Acorn BBC Micro, an Acorn Archimedes, which was my first ARM based home computer, and various DOS machines until I ended up on Windows, then a Macintosh. I have a large collection of classic 8- and 16-bit micros from the 1980s. More recently, I have been playing around with Linux on x86 and ARM platforms for media playback, TV reception, VOIP PBX, and home automation, and have started playing with ESXi VMs. I also have an unRAID server that I built, which stores all my media.

I was attracted to the ODROID platform for its performance and price. I initially bought a U2, since it generated a lot of buzz in Kodi circles (I'm a moderator on the Kodi forums), which was a little disappointing in the end. The U2 didn't quite get the support it needed to be a great platform, even though the hardware should have allowed it to fly. My U2 was a fun toy, but it never really gets much use. The C1 and C1+ were my next purchases, and they showed promise. However, the C2 was the game changer for me. My C2 is one of my main 2160/50p HEVC players running LibreElec. Forum user @wrxtasy and the team developing LibreElec and Kodi for the AMLogic S905 platform deserve a lot of credit for their hard work in getting Kodi to such a great point on the ODROID platform.

Stephen enjoys reading crime fiction

I mainly use my C2 for running LibreElec to run Kodi. The lack of a modern kernel, which I know is coming, means that running my C1/C1+s as TV Headend backends is a pain because of limited kernel support of modern DVB-T2 tuners, which would be my ideal task for them. I run the C1s as secondary media players. I have an x86 box running as my main TV Headend server at the moment.

The ODROID-C2 is my favorite ODROID because it offers awesome media player functionality at a fantastic price. The eMMC storage is nice and fast. It would be nice to see integrated 802.11ac WiFi and Bluetooth 4.0 or higher, along with a Displayport output capable of retina resolutions. HDMI is great, but running an iPad Retina screen via an eDP to DP converter, which is a trivial bit of hardware and widely available, would allow the use of cheap 9.7" retina displays, which would make an awesome "not quite tablet". USB 3.0 support would also be fantastic for high speed storage bandwidth. Effectively, it would be great to see a C2 / XU4 hybrid, if such a SoC exists!



The ZX81 computer kit was an amazing machine when it was first introduced, and got Stephen started with computers

I'd also like to see up-to-date kernels. The Raspberry Pi and x86 platforms are more flexible than the ODROID because of this. I love my ODROIDs, but an up-to-date kernel with driver support for new devices would make them even better.

In my free time, I enjoy learning languages, watching TV, reading crime fiction, and traveling. I've been fortunate to be able to travel widely in my job, and really enjoy visiting new places.

My advice for new programmers is to just jump in and buy a cheap ARM computer like the ODROID-C1+ or C2, or a Raspberry Pi, since its educational support is second-to-none, and learn about Linux. Afterwards, branch out to learning Python. You should also use Google wisely, since you can learn a lot by seeing how other people have solved similar problems.