Touchscreen Driver installation • Community images for ODROID-CI

ODROID Magazine

Year Two
Issue #18
Jun 2015

Desktop Case

The coolest way to show off your ODROID classic gaming computer

The original ODROID

- ODROID-U3 Car PC
- Using a CPLD as a programmable level shifter
- UltraStar Deluxe Karaoke
- Nintendo 64 Gaming: part 2

What we stand for.

We strive to symbolize the edge of technology,
future, youth, humanity, and engineering.

Our philosophy is based on Developers.
And our efforts to keep close relationships with
developers around the world.

For that, you can always count on having the quality
and sophistication that is the hallmark of our products.

Simple, modern and distinctive.
So you can have the best to accomplish
everything you can dream of.

HARDKERNEL

Pollin ELECTRONIC

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

Address: Max-Pollin-Straße 1
85104 Pförring Germany

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

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

EDITORIAL

Dongjin, one of the Hardkernel engineers, decided to build a case for his ODROID that looks like the Macintosh Plus model from his youth. We think he did a great job in creating a modern, futuristic version of that classic computer. Since we did a feature article on building a Truck PC last year, we've had requests for a similar article for sedan owners. In response, Belov replaced the standard built-in electronics gear in his Opel Astra using an ODROID-U3 to provide both music and map services. Venkat also presents a cool project for your car that allows you to obtain detailed vehicle information via Bluetooth, download it to an ODROID, and plot the results using Google Earth. Tobias continues his Nintendo 64 series with reviews of several very popular N64 games, Nanik details the process of creating a custom service for Android, Bo teaches us how to install touchscreen drivers on an ODROID-C1, and Carsten presents his Guzunty Pi project of building an inexpensive UART console. We also find out how to turn an ODROID into a karaoke machine, play Tekken 6 in HD resolution, install some convenient community pre-built disk images, and much more!

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/IyplmXs.
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

HARDKERNEL'S EXCLUSIVE NORTH AMERICAN DISTRIBUTOR

ameriDroid
High-Performance Embedded Computers

All Hardkernel products in stock at AmeriDroid.com


USB GPS MODULE
$26.95


ODROID-C1
$36.95

ODROID-VU
$119.95


C1 3.2 INCH TOUCHSCREEN DISPLAY SHIELD
$26.95

ODROID Magazine

Rob Roy, Chief Editor

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

Bo Lechnowsky, Editor

I am President of Respectech, Inc., a technology consultancy in Ukiah, CA, USA that I founded in 2001. From my background in electronics and computer programming, I manage a team of technologists, plus develop custom solutions for companies ranging from small businesses to worldwide corporations. ODROIDs are one of the weapons in my arsenal for tackling these projects. My favorite development languages are Rebol and Red, both of which run fabulously on ARM-based systems like the ODROID-U3. Regarding hobbies, if you need some, I'd be happy to give you some of mine as I have too many. That would help me to have more time to spend with my wonderful wife of 23 years and my four beautiful children.

Bruno Doiche, Senior Art Editor

Nothing gets into our Senior Art Editor's blood as the rush of long nights doing and re-doing the cover of the magazine. As a Gemini, he actually enjoys it, even when his family strives to get his attention and figure out what he wants for his birthday. How old is Bruno? 100101 years old!

Nicole Scott, Art Editor

I'm a Digital Strategist and Transmedia Producer specializing in online optimization and inbound marketing strategies, social media directing, and media production for print, web, video, and film. Managing multiple accounts with agencies and filmmakers, from Analytics and Adwords to video editing and DVD authoring. I own an ODROID-U3 which I use to run a sandbox web server, live in the California Bay Area, and enjoy hiking, camping and playing music. Visit my web page at http://www.nicolecscott.com.

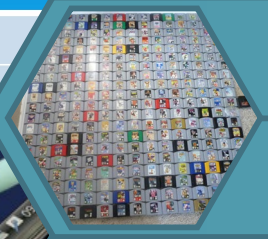
James LeFevour, Art Editor

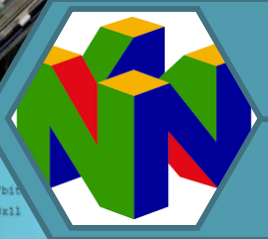
I am a Digital Media Specialist who is also enjoying freelance work in social network marketing and website administration. The more I learn about ODROID capabilities, the more excited I am to try new things I'm learning about. Being a transplant to San Diego from the Midwest, I am still quite enamored 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.

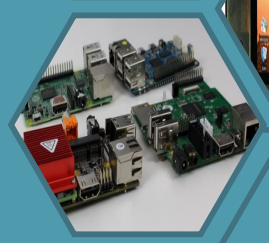
Manuel Adamuz, Spanish Editor

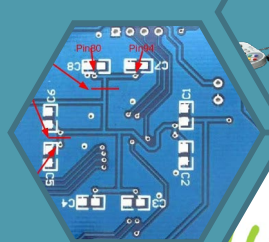
I am 31 years old and live in Seville, Spain, and was born in Granada. I am married to a wonderful woman and have a child. A few years ago I worked as a computer technician and programmer, but my current job is related to quality management and information technology: ISO 9001, ISO 27001, and ISO 20000. 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.

INDEX







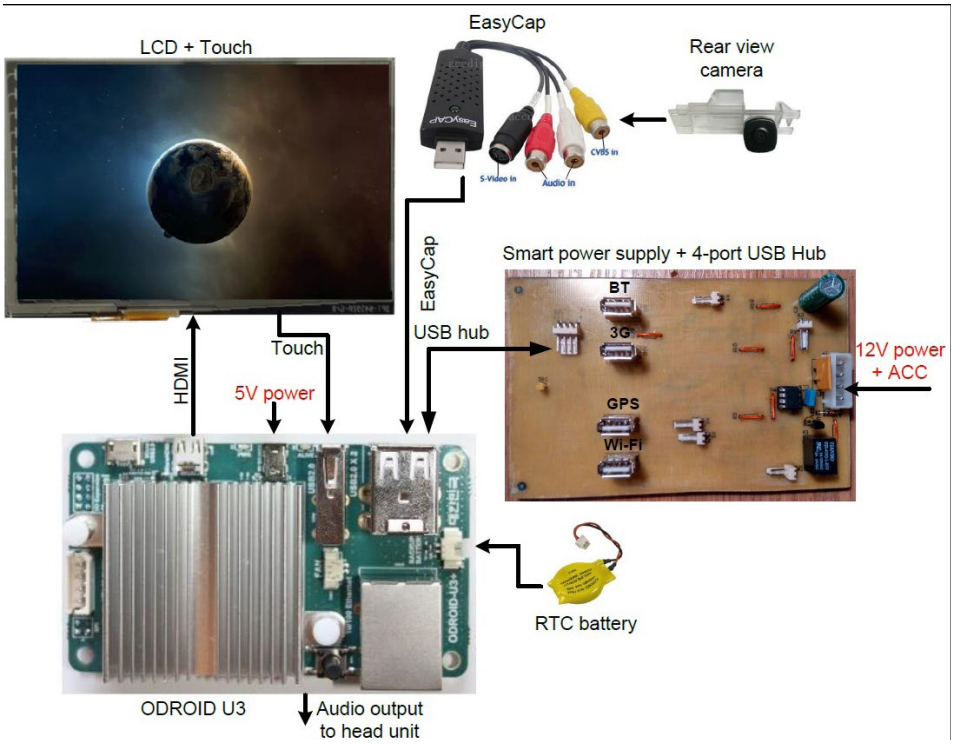


ODROID-U3 CAR PC

REPLACING THE STANDARD FACTORY EQUIPMENT

by Belov Vitaly

My car, an Opel Astra H, came factory-equipped with a radio and CD MP3 player as well as a black-and-white 4-inch graphic display. I was not satisfied with the lack of USB input for connecting flash drives with MP3 music, and I had to use a smartphone for navigation, which constantly requires a battery charge and can only be placed on the windshield. At first, I bought a commercial pre-built device to replace the head unit, which was very uncomfortable, because the screen of the device was too low, at the level of my stomach. Another option was to mount a 7 inch tablet, such as the Nexus 7, but it didn't fit into the frame, since the standard Astra H only has space for a 6.5-inch LCD, which was too small for me. I then decided to build an Android CarPC and install it in place of the black and white display.



Component diagram

Completed Car PC installation

I needed the following functionality from my CarPC:

- Navigation
- Mp3 player
- Internet radio
- Video player
- Rear view camera

I chose an ODROID-U3 as the basis since it is an inexpensive but powerful computer with excellent performance and plenty of memory.

Components

- ODROID-U3 with 8Gb eMMC
- RTC (Real Time Clock) battery
- 7" 1280x800 IPS LCD with 5 points capacitive multitouch
- Frame for 7" LCD (my own development, 3D printing)
- Bluetooth USB dongle
- Wi-Fi USB dongle
- Atmel ATTiny
- HUAWEI E1550 3G USB modem
- GPS/GLONASS USB Holux

M-215+
- EasyCap USB with an STK1160 chip for the rear view camera
- OBD2 Bluetooth adapter
- Rear view camera
- Power supply DC to DC 12V to 5V with 4-port USB hub
- ELM327 OBD2 adapter

I was not able to find a ready-made frame for the 7-inch LCD, so I had to do it myself. The frame layout for 7" LCD was created in SolidWorks and then printed on a 3D printer. I then plastered, sanded and painted the frame in matte black, which fit perfectly, giving a visible area of 152.5x91.5mm. The display was attached to the frame using 0.5mm double-sided tape.

I chose a ChalkElec 7-inch LCD with a resolution of 1280x800 pixels and 5 points multi-touch The LCD is very bright with good contrast, and the image is clearly visible in sunlight. The ODROID-U3 has a micro HDMI output, and the screen from ChalkElec has a mini HDMI input. Since this cable is not readily available, I had to make it myself. The multi-touch cable is connected to the ODROID-U3 via USB port, with single-touch capacity working out of the box. To support 5-point multi-touch, you need to compile the kernel with support for ChalkBoard Touch, which is detailed in the "Giant Tablet" article in the February 2014 issue of ODROID Magazine.

Smart Power Supply

The ODROID-U3 requires 5 volts, and there are a large number of 12V to 5V converters with a variety of power available commercially. However, I wanted to automate turning the CarPC on and off depending on the position of the ignition key, which is indicated by the ACC signal from the car, requiring a custom power supply. I chose the chip LT1374 for the 12V to 5V converter. It is quite simple, and consumes very little energy in standby mode, yet delivers current up to 2A and has on/off control input. The cheapest microcontroller (MC) ATTINY13 was used in order to automate the power supply with the ACC control signal.

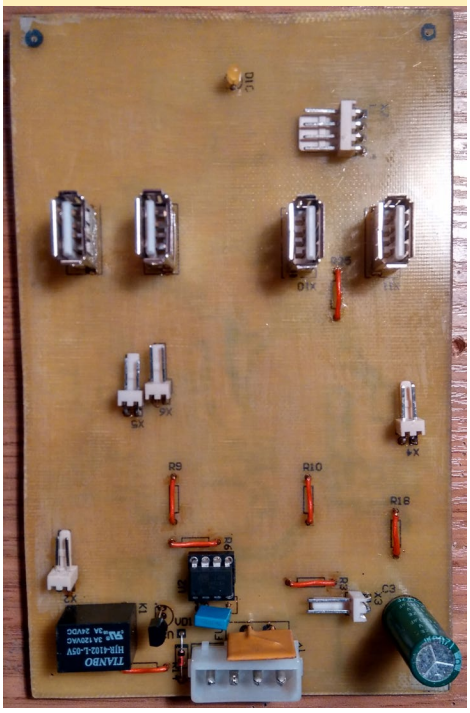
LCD frame components

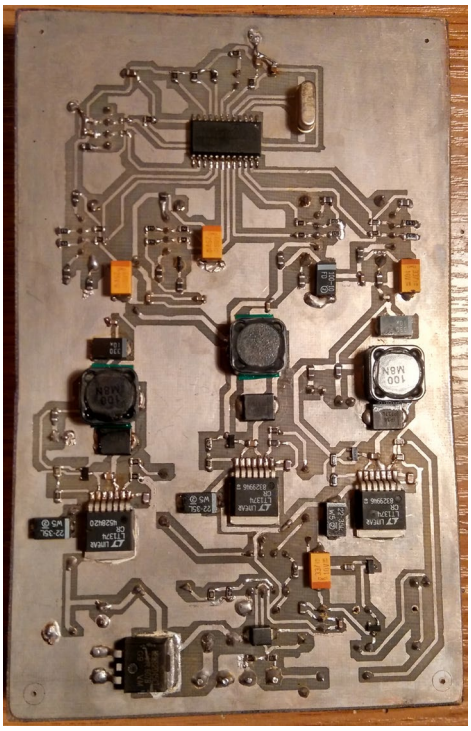
Assembled LCD Frame

Algorithm

- On first start, the microcontroller engages and monitors the ACC signal.
- when the ACC signal is on, the periphery is powered, and after a couple of seconds, the ODROID-U3 is powered.
- when the ACC signal is off, the "Power" button pressing is emulated, the ODROID-U3 goes to "Sleep mode", and "Airplane mode" is turned on.
- when the ACC signal is off, after 1 second, the "Power" button pressing is emulated, the ODROID-U3 wakes up, and "Airplane mode" is turned off.
- when the ACC signal is off for more than 30 seconds, the car battery is monitored so that at less than 11.5V, everything depowered except the microcontroller.
- when the ACC signal is off for more than 14 hours, the ODROID-U3 is depowered.

Smart Power Supply front and rear view





USB Hub

An assembled USB hub is also mounted on the power supply. In order to work in the car at low temperatures during the Russian winter, I selected the industrial chip AT43301 (ATMEL), which works at temperatures as low as -40˚C. To filter out noise on all lines of power, ferrite beads (FB) were installed on the USB hub. Without ferrite beads, the chip on the integrated USB hub on the ODROID-U3 board will occasionally freeze.

Android and software

The latest Android 4.4.4 KitKat 4.5 image from Hardkernel supports many devices automatically, but does not support ChalkBoard Touch and EasyCap, which makes it necessary to compile a kernel for use with these devices. However, there is a simpler solution. Thanks to ODROID forum member @voodik, there is an Android 4.4.4 KitKat Cyanogenmod 11.0 community image which already supports ChalkBoard Touch and EasyCap, which may be downloaded from http://bit.ly/1Lc2nWW.

For correct operation of the GPS receiver, you must specify the correct baud rate of the receiver and number of USB port in the file /system/build.prop. For my ODROID-U3, the port is ttyUSB3, and the speed is 4800:

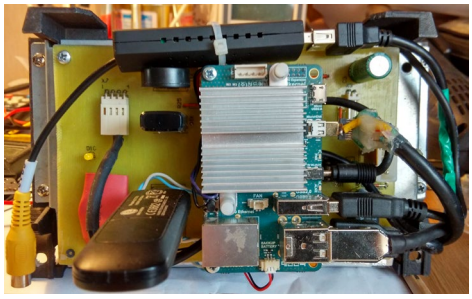
```
ro.kernel.android.gps=ttyUSBx
ro.kernel.android.gps.speed=xxxx
```

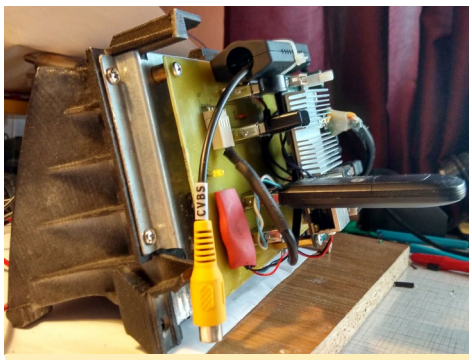
Once the system is installed, you must install the GoogleApps Installer application in order to use the Google Play Store. For the home screen, I chose the app called BigLauncher, which is designed for elderly or visually impaired people, since it has large icons and is very well suited for use in the car. To listen to MP3 music, I installed PowerAmp, and PC Radio for Internet radio. For viewing photos, I chose Quick Pic, and for playing videos, I used MX Player. For connecting the mobile phone to the CarPC, I installed Tablet Talk, and configured Torque Pro for communication between the on-board car computer using an OBD2 adapter. To reduce energy consumption in sleep mode, I also installed the Automate It app. After disabling the ACC signal, a power button press is emulated by Automate It, along with toggling Airplane mode.

Android Home Screen

Installation

To mount the screen, I glued the LCD to the frame using a thin double-sided tape. I then secured the power supply and ODROID-U3 using a custom metal bracket, and connected all USB devices. For power, I used individual 12V wires from the car battery to my CarPC. The audio output of the ODROID-U3 was then connected to the AUX input of the head unit of the car, and I mounted the GPS receiver in the side of the rack windshield.





View of assembled CarPC before installation in the vehicle

The CarPC connects to the Internet via a hotspot from my mobile phone. Also, just in case the phone runs out of battery, a 3G USB dongle was connected to the CarPC.

Software

The code for Attiny13 was written in CodeVisionAVR, which is available at http://bit.ly/1RY4G4s. To flash the Attiny13 with the controller code, create a new blank project in CodeVisionAVR, then download the code from http://pastebin.ca/3002845 and copy it into the new project. Compile the code in order to produce a HEX firmware file, then flash the HEX file to the Attiny13 using Arduino software.

Everything now works great in my car. Using an ODROID-U3 board, I created a powerful and fast CarPC, which improved upon the original capabilities of the factory-installed system.

The combination of Android and your car makes a cost-effective alternative to the stock factory equipment

With an ODROID-U3 Car PC, you will be the envy of all of your friends

OBDGPS LOGGER

COMBINE OBDII AND GPS DATA FOR COMPLETE VEHICLE TRACKING

by Venkat Bommakanti

In the May 2015 issue, I introduced the powerful combination of an ODROID-C1 and a USB GPS module in order to track vehicles using OpenGTS. What if we could further enhance that combination by adding vehicle diagnostics to the mix? Well, On-Board Diagnostics (OBDII) does exactly that! This article outlines how to get both the USB GPS and a bluetooth OBDII adapter working synchronously with the C1 using the open-source package called OBDGPSLogger, which may be used with vehicles made after 1999.

OBDGPSLogger is a C-based command-line solution designed to run on Linux with the acquired data saved to a local SQLite database. It does not come with any native database management front-end. However, to assist with the management of the OBD and GPS data, this example uses a browser-based management option utilizing Nginx as the web server, PHP as the scripting engine, and phpLiteAdmin as the SQLite database administration frontend.

It is possible to migrate this to a MySQL-based solution or use other utilities like pyOBD, which is a Python-based open-source utility for acquiring OBD data, but only the basic setup will be detailed here. Refer to the May 2015 OpenGTS article for instructions on setting up, configuring, and validating the USB GPS adapter.

Requirements

1. An ODROID-C1, although these steps can also apply to a more powerful ODROID system.
2. Required C1 accessories such as an HDMI cable, a CAT 5E+ ethernet cable or WIFI 3 adapter, a Bluetooth adapter module 2 (BT 4.0+), a power supply unit, an RTC battery, and an HDMI-compatible monitor, ODROID-VU or 3.2" touchscreen.
3. A 16GB+ eMMC 5.0 module or microSD card with the latest Lubuntu desktop image, and an SD card reader/writer.
4. A USB GPS module from Hardkernel, available at http://bit.ly/1EPERhm.
5. A compatible OBDII (v1.5+) Bluetooth adapter with an ELM327 interface, such as the one produced by Panlong.
6. A network where the device has access to the internet and the ODROID forums.
7. OBDGPSLogger software (Version: 0.16) gpsd and gpsd-client.
8. Networked access to the C1 via utilities like PuTTY, FileZilla, TightVNC Viewer (MS Windows 7+), or Terminal (Mac, Linux), from a testing desktop.
9. LiPo battery packs with at least two (2) USB ports providing 2A+ each: one for the VU (if 3.2" Touchscreen not used) and another for the C1 itself, along with C1-compatible USB-DC plug cables, available from Hardkernel.

Figure 1: Assembled C1, GPS, OBDII, Touchscreen and Battery portable setup

Install Lubuntu

Install the latest C1 image onto the eMMC module or SD card and attach it to the ODROID. With the display attached, boot up the system. Run the ODROID Utility and set the display resolution, then reboot.

Expand the installation partition to use all of the available drive space by selecting the "Resize your root partition" option. Reboot and re-run the ODROID Utility to configure and update all remaining relevant aspects of the system, and reboot again.

Ensure you are logged in as the default odroid user, unless otherwise specified. Type the following commands in a Terminal window in order to update the operating system files, system kernel, and related applications:

```
$ sudo apt-get autoremove && sudo apt-get update
$ sudo apt-get dist-upgrade && sudo apt-get upgrade
$ sudo apt-get install linux-image-c1
```

Shutdown the ODROID, attach all of the accessories and cables, including the GPS and Bluetooth adapters, then reboot. Check the system version from a Terminal window using the following command to ensure you have the latest version:

```
$ uname -a
Linux odroid 3.10.75-84 #1 SMP PREEMPT Sat Apr 25 18:33:08 BRT 2015 armv7l armv7l armv7l GNU/Linux
```

Setup Bluetooth

Install additional utilities using the following commands:

```
$ sudo apt-get install bluez-dbg bluez-hcidump bluez-utils bluez-tools
$ sudo apt-get install bluewho blueman python-bluetooth
```

Then, check the USB information related to the adapters:

```
$ lsusb
…
Bus 001 Device 005: ID 0b05:17cb ASUSTek Computer, Inc.
Bus 001 Device 004: ID 1546:01a6 U-Blox AG
…
```

Note that I used an ODROID-compatible ASUS Bluetooth 4.0 adapter in place of the Hardkernel model. Check the Bluetooth adapter's support for additional features such as RFCOMM protocol by examining the dmesg logs:

```
$ dmesg | grep Blue
[     0.851848@0] Bluetooth: Core ver 2.16
[     0.859721@0] Bluetooth: HCI device and connection manager initialized
[     0.866240@0] Bluetooth: HCI socket layer initialized
[     0.871245@0] Bluetooth: L2CAP socket layer initialized
[     0.876447@0] Bluetooth: SCO socket layer initialized
[     1.429422@2] Bluetooth: HCI UART driver ver 2.2
[     1.433876@2] Bluetooth: HCI H4 protocol initialized
[     1.438828@2] Bluetooth: HCI BCSP protocol initialized
[     1.443919@2] Bluetooth: HCILL protocol initialized
[     1.448782@2] Bluetooth: HCI-ATH3K protocol initialized
[     1.453877@2] Bluetooth: HCI Three-wire UART (H5) protocol initialized
[     3.236424@2] Bluetooth: bt-wake_control_init Driver Ver 1.1
[     3.366366@2] Bluetooth: RF-COMM TTY layer initialized
[     3.371156@2] Bluetooth: RF-COMM socket layer initialized
[     3.376392@2] Bluetooth: RF-COMM ver 1.11
[     3.380308@2] Bluetooth: BNEP (Ethernet Emulation) ver 1.3
[     3.385744@2] Bluetooth: BNEP filters: protocol multicast
[     3.39789S@2] Bluetooth: BNEP socket layer initialized
[     3.402975@2] Bluetooth: HIDP (Human Interface Emulation) ver 1.2
[     3.409060@2] Bluetooth: HIDP socket layer initialized
```

Check the installed Bluetooth modules:

```
$ dpkg -l | grep blue
ii  blueman                ...
armhf  Graphical bluetooth manager
ii  bluez                  ...
armhf  Bluetooth tools and daemons
ii  bluez-alsa:armhf       ...
armhf  Bluetooth ALSA support
ii  bluez-cups             ...
armhf  Bluetooth printer driver for CUPS
ii  libbluetooth-dev       ...
armhf  Dev.  files for BlueZ Linux Bluetooth lib
ii  libbluetooth3:armhf    ...
armhf  Library to use the BlueZ Linux BT stack
ii  libgnome-bluetooth11   ...
armhf  GNOME Bluetooth tools - support library
```

Check for the presence of the Bluetooth device, which will be useful for connection configuration:

```
$ hcitool dev
Devices:
    hci0    00:02:72:CC:F4:CE

$ hciconfig
hci0:    Type: BR/EDR  Bus: USB
    BD Address: 00:02:72:CC:F4:CE ACL MTU: 1021:8  SCO MTU: 64:1
    UP RUNNING PSCAN
    RX bytes:583 acl:0 sco:0 events:33 errors:0
    TX bytes:898 acl:0 sco:0 commands:33 errors:0

$ sudo rfkill list all
0: hci0: Bluetooth
    Soft blocked: no
    Hard blocked: no
```

Configure Bluetooth

From the lubuntu desktop, launch the Bluetooth Manager configuration utility as shown in Figure 2. Then, select the Preferences menu item to configure the application, according to Figure 3. Using Figure 4 as a reference, update the friendly name of the Bluetooth adapter to a meaningful one like "c1-1-0". Make the device always visible for other Bluetooth devices to scan and find it. Save

the configuration and reboot.

Figure 2 - Launch Bluetooth Manager

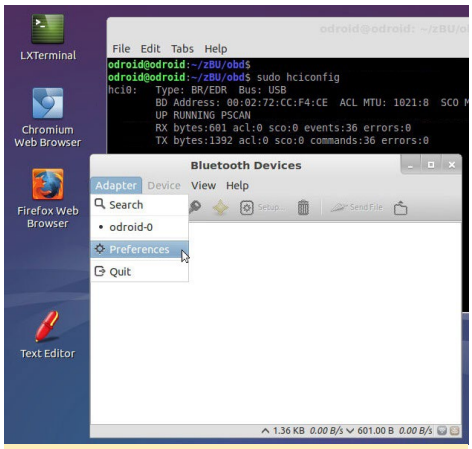


Figure 3 - Bluetooth Manager configuration

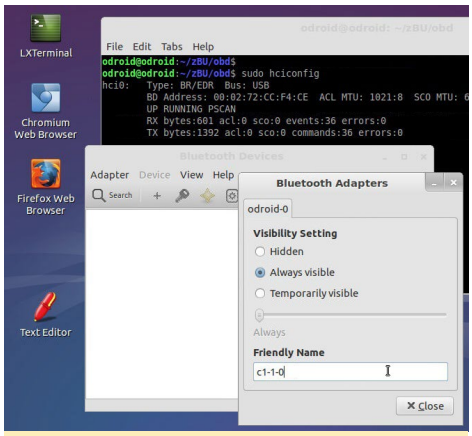


Figure 4 - Bluetooth Manager configuration update

Testing Bluetooth

If necessary, you can use the command-line Bluetooth packet analyzer/sniffer to analyze Bluetooth traffic for debugging purposes:

```
$ sudo hcidump
HCI sniffer - Bluetooth packet
analyzer ver 2.5
device: hci0 snap_len: 1500 fil-
ter: 0xffffffff
< HCI Command: Write Class of De-
vice (0x03|0x0024) plen 3
      class 0x700100
> HCI Event: Command Complete
(0x0e) plen 4
      Write Class of Device
(0x03|0x0024) ncmd 1
      status 0x00
< HCI Command: Write Extended
Inquiry Response (0x03|0x0052)
plen 241
      fec 0x00
      Complete local name: 'c1-1-
0'
      TX power level: 0
      Complete service class-
es: 0x112d 0x1112 0x111f 0x111e
0x110c 0x110e 0x1105
> HCI Event: Command Complete
(0x0e) plen 4
      Write Extended Inquiry Re-
sponse (0x03|0x0052) ncmd 1
      status 0x00
< HCI Command: Write Extended
Inquiry Response (0x03|0x0052)
plen 241
      fec 0x00
      Complete local name: 'c1-1-
0'
      TX power level: 0
      Complete service class-
es: 0x112d 0x1112 0x111f 0x111e
0x110c 0x110e ...
> HCI Event: Command Complete
(0x0e) plen 4
      Write Extended Inquiry Re-
sponse (0x03|0x0052) ncmd 1
      status 0x00
...
```

Another useful tool is Wireshark to give a graphic view of the snooped data, which may be installed by typing the following command into any Terminal window:

```
$ sudo apt-get install wireshark
```

After installation, you can launch Wireshark version 1.10.6 following the steps illustrated in Figure 5. The welcome screen should appear as seen in Figure 6, indicating that the Bluetooth adapter on the C1 has been detected.

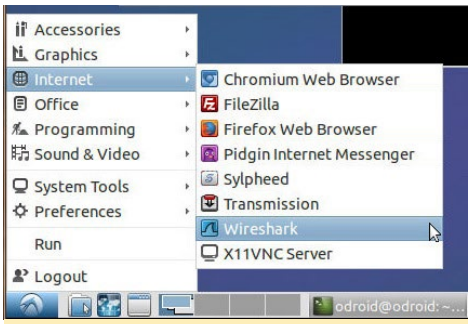


Figure 5 - Launch Wireshark

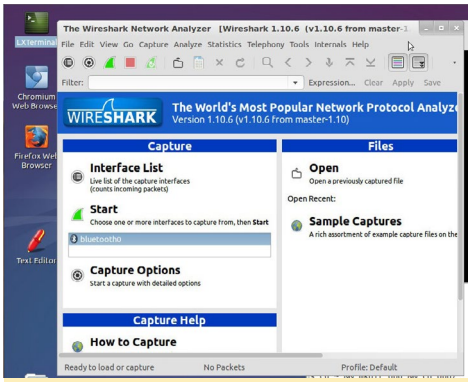


Figure 6 - Wireshark welcome screen with Bluetooth adapter detected

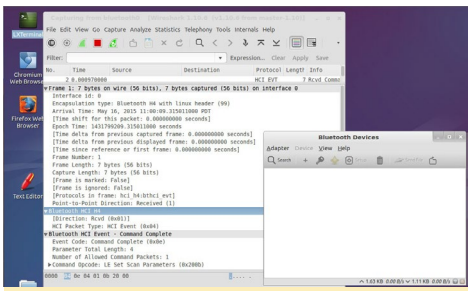


Figure 7 - Wireshark with snooped Bluetooth traffic

Click on the green shark fin icon to request start of a capture, then launch the Bluetooth Manager application and click on the Search button after the application has loaded. You should immediately see all of the snooped Bluetooth traffic and related information appear in the Wireshark application window, as illustrated in Figure 7.

Install gpsd

Install gpsd and relevant utilities using the following command. For detailed information on configuration, testing and usage, please refer to the May 2015 OpenGTS article.

```
$ sudo apt-get install gpsd gpsd-
clients && sudo reboot
```

Install web server tools

OBDGPSLogger stores the collected OBDII and GPS data in a local SQLite3 database. It does not include a native database management tool. To help with possible database management needs, I have also included the installation of nginx web server and PHP along with phpLiteAdmin in order to display the SQLite3 administrative GUI. First, install the following:

```
$ sudo apt-get install nginx-full
sqlite3
$ sudo apt-get install autoconf
automake autotools-dev libtool
curl
$ sudo apt-get install libcurl4-
openssl-dev lbzip2
$ sudo apt-get install php5 php5-
dev php5-cgi php5-fpm php5-curl
php5-gd
$ sudo apt-get install php5-
sqlite php5-gmp php5-imagick
php5-imap php5-intl
$ sudo apt-get install php5-ldap
php5-mcrypt libmcrypt-dev php-
xml-parser
$ sudo apt-get install php5-xsl
php-apc
```

Then, update nginx configuration to enable PHP5 support:

```
$ sudo cd /etc/nginx/sites-avail-
able
$ sudo cp default default-orig
$ sudo medit default
```

Update the default configuration:

```
...
# our php-handler - add this
upstream php-handler {
    server unix:/var/run/php5-
fpm.sock;
}

server {
    listen 80 default_server;
    listen [::]:80 default_server
ipv6only=on;

    root /usr/share/nginx/html;

    # try php file execution first
    index index.php index.htm;

    # Make site accessible from
http://localhost/
    server_name <your-C1's-ip-
address>;

    # set max upload size
    client_max_body_size     10G;
    fastcgi_buffers          64
4K;
    client_body_buffer_size 2M;

    # setup calendar, contact,
webdav options
    rewrite ^/caldav(.*)$ /re-
mote.php/caldav$1 redirect;
    rewrite ^/carddav(.*)$ /re-
mote.php/carddav$1 redirect;
    rewrite ^/webdav(.*)$ /re-
mote.php/webdav$1 redirect;

    location = /robots.txt {
         allow all;
         log_not_found off;
         access_log off;
    }

    location / {
        # First attempt to serve
request as file, then
        # as directory, then fall
back to displaying a 404.
        try_files $uri $uri/ in-
dex.php;

        # The following 2 rules
are only needed with webfinger
        rewrite ^/.well-
known/host-meta /public.
php?service=host-meta last;
        rewrite ^/.well-
known/host-meta.json /public.
php?service=host-meta-json last;
        rewrite ^/.well-known/
carddav /remote.php/carddav/ re-
direct;
        rewrite ^/.well-known/
caldav /remote.php/caldav/ redi-
rect;
        rewrite ^(/core/doc/
[^\/]+/)$ $1/index.html;
    }

    # redirect server error pages
to the static pages
    error_page 404 /404.html;
    error_page 500 502 503 504
/50x.html;
    location = /50x.html {
       root /usr/share/nginx/
html;
    }

     # pass the PHP scripts to
FastCGI server listening on fpm-
socket
     location ~ \.php(?:$|/) {
        fastcgi_split_path_info
^(.+\.php)(/.+)$;
        include fastcgi_params;
        fastcgi_param SCRIPT_
FILENAME $document_root$fastcgi_
script_name;
        # $fastcgi_path_info
parse fails in latest php5-fpm.
disable it.
        # fastcgi_param PATH_INFO
$fastcgi_path_info;
        fastcgi_pass php-handler;
        fastcgi_read_timeout 600;
     }
}
...
```

Next, update the php5-fpm configuration using the following commands:

```
$ cd /etc/php5/fpm/pool.d/
$ sudo cp www.conf www.conf-orig
$ sudo medit www.conf
```

Add the following socket configuration to match the nginx socket configuration:

```
...
listen = /var/run/php5-fpm.sock

Enhance file execution security by
setting the following flags in the
php5 config file:
$ sudo medit /etc/php5/fpm/php.
ini

Set these options:
    cgi.fix_pathinfo=0
    display_errors = On
    display_startup_errors = On
    output_buffering = 4096
    default_socket_timeout = 600
...
```

After modifying each file, save the changes. Then, download phpLiteAdmin version 1.9.5 from http://bit.ly/1HHIJAJ and prepare for the build:

```
$ cd ~/obd && mkdir pla && cd pla
$ mv ~/Downloads/phpliteAdmin_v1-
9-5.zip .
$ unzip phpliteAdmin_v1-9-5.zip
```

Refer to the README file for setup tips:

```
$ cat README.txt
```

Update phpLiteAdmin config using data relevant to your setup:

```
$ cp phpliteadmin.config.sample.
php phpliteadmin.config.php
$ medit phpliteadmin.config.php
```

Change the following data:

```
...
    $password = 'odroid';

    $directory = '/home/odroid/
obd/obdgpslogger/data';

    $databases = array(
       array(
               'path'=> 'obdg-
pslogger.db',
               'name'=> 'OBDII-
GPS Logger'
       ),
    )
...
```

Then, move the administrative application and related configuration files to the appropriate place, and set its execution privileges. Reboot after saving the file.

```
$ sudo cp phpliteadmin.config.php
/usr/share/nginx/html
$ sudo cp phpliteadmin.php /usr/
share/nginx/html
$ chmod 755 phpliteadmin.php

$ cd /usr/share/nginx/html
$ ls -lsa
...
  4 -rw-r--r-- 1 root root   537
Mar  4  2014 50x.html
  4 -rw-r--r-- 1 root root   612
Mar  4  2014 index.html
  4 -rw-r--r-- 1 root root   2691
May 17 10:49 phpliteadmin.config.
php
220 -rwxrwxr-x 1 root root 222859
May 17 10:49 phpliteadmin.php
```

Build OBDGPSLogger

Install additional prerequisites for OBDGPSLogger:

```
$ sudo apt-get install libx11-
dev libxft2-dev libgps-dev zlibc
zlib1g zlib1g-dev
$ sudo apt-get install libfltk1.3-
dev fluid libftdi1 libftdi-dev
subversion
```

Download the OBDGPSLogger code using the commands:

```
$ cd ~ && mkdir obd && cd obd/
$ svn co svn://svn.icculus.org/
obdgpslogger/trunk obdgpslogger
$ cd obdgpslogger/obd/obdgpslog-
ger/src/logger
```

The OBDGPSLogger code needs to be modified in order to work with the latest gpsd services. Apply the following patch to the source code:

```
~/obd/obdgpslogger/src/logger/
gpscomm.c

29c29,32
<     struct gps_data_t *g = gps_
open(server,port);
---
>     int rc;
>     struct gps_data_t *g =
NULL;
>
>     g = malloc(sizeof(struct
gps_data_t));
31a35,40
>
>     rc = gps_open(server,port,
g);
>     if(rc != 0) {
>      free(g);
>      return NULL;
>     }
61c70
<             gps_poll(g);
---
>             gps_read(g);
```

It is now ready to be built. Resume the process with the following commands:

```
$ cd ~/obd/obdgpslogger
$ mkdir build
$ cd build
$ cmake ..
$ make
$ sudo make install
```

Verify the installation using the command:

```
$ obdgpslogger -v
Version: 0.16
```

Pair the C1 and OBDII adapter

Shut down the ODROID, park your car at a safe place, and turn off the ignition. Attach the OBDII adapter to the appropriate port in your car, which is usually located a few feet from the steering wheel, close to the foot-brake or accelerator. Let it go through its initialization process, then place the C1 on the passenger seat next to you. It should be kept within the recommended 10-foot range from the OBDII adapter (the closer, the better).

Not all OBDII adapters are compatible with ODROIDs. Before purchasing one, research the adapter well to ensure that it is supported under Linux. I took a chance with my $10 adapter, and was

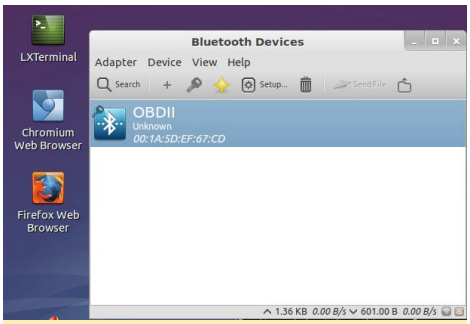


Figure 8 - OBDII adapter detected by the ODROID-C1

fortunate that it worked.

Start the car, power up the C1, and start the Bluetooth Manager application. Click the Search button to scan for the OBDII adapter. After a few moments, you should see the result of the discovery process as shown in Figure 8.

If it is not detected, remove and insert the OBDII adapter a couple of times and retry the Search. If it still fails, you can test the Bluetooth connectivity with a smartphone. Once it is discovered, select the Serial port connection as shown in Figure 9. Attempt to pair the C1 and

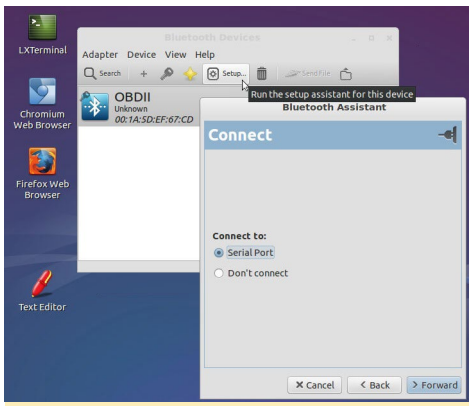


Figure 9 - Serial port setup

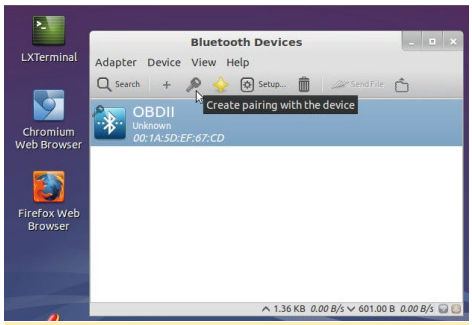


Figure 10 - Pairing the C1 and OBDII adapter using Bluetooth

the OBDII adapter by clicking on the Key button as shown in Figure 10:

You will be prompted for a 4-digit key to complete the pairing process, which should be available from the OBDII adapter vendor. It could be one of the commonly used keys, such as 0000 or 1234. Research the Internet for the key code if it is not provided by the vendor.

To ensure that the OBDGPSLogger application would work correctly, check to see if the USB device profile was created on the C1. The presence of these two entries indicates that the system is working well.

```
$ ls -lsa /dev/rf*
0 crw-rw---- 1 root dialout 216,
0  Dec 31 16:22 /dev/rfcomm0
0 crw-r--r-- 1 root root     10,
63 Dec 31 16:00 /dev/rfkill
```

Acquire and plot data

Create a placeholder for the data to be captured, then start the OBDGPSLogger application, making sure to specify the serial port noted above. Inspect the output to ensure that OBDGPSLogger is ready to capture both the GPS and OBD data successfully:

```
$ cd ~/obdgpslogger
$ mkdir data && cd data

$ sudo$  obdgpslogger -s /dev/
rfcomm0
Opening serial port /dev/rfcomm0,
this can take a while
Successfully connected to serial
port.  Will log obd data
Successfully connected to gpsd.
Will log gps data
Creating a new trip
GPS acquisition complete
```

Now, take your C1 setup for a ride around the block! The OBDGPSLogger should start capturing the vehicle data and storing it in the SQLite database. After the trip has completed, exit the OBDGPSLogger application by pressing Control-C, then go home to start the plotting process. Export the collected data in KML format to be used with Google Maps and Google Earth:

```
$ cd ~/obdgpslogger/data
$ obd2kml -d obdgpslogger-vb.db
-o obdgpslogger-vb.kml
```

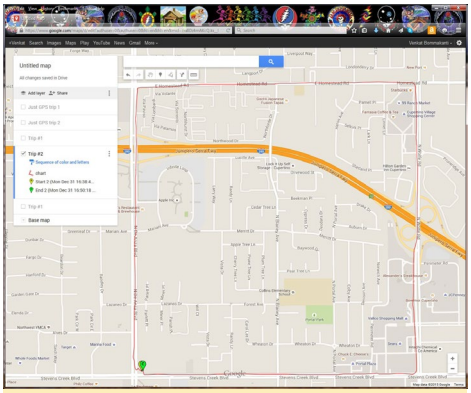


Figure 11 - KML data in Google Maps

```
$ ls -lsa *.kml
276 -rw-rw-r-- 1 odroid odroid
278707 May 16 22:06 obdgpslogger-
vb.kml
```

Then, upload the KML file to Google Maps, as shown in Figure 11.

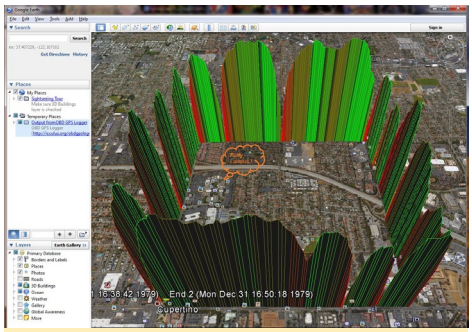


Figure 12 - KML data in Google Earth

- Log in to your Google Account, and go to http://maps.google.com
- Click on My Maps
- Click Create a new map
- Add a Title and Description
- Click Import
- Click Choose file, select the above .kml to upload, and then click Upload from file

Next, install Google Earth on your Windows 7+ or Macintosh system. Transfer the .kml file from the C1 to your system with Google Earth. Import the .kml file and display it, as shown in Figure 12.

Note that, while the GPS data is used to plot the course of the ride, the OBD data is used to plot the car's fuel efficiency in miles per gallon. The red parts correspond to low fuel efficiency going from 0 mph to posted maximum of 35 mph. The green parts correspond to the efficient segments. The dips correspond to the car stopping at all the stop signs in between.

Do you recognize the world-famous fruity campus that was encircled? There's a clue at bottom left of picture!

The data can also be exported in the .csv format, and imported in MS Excel or compatible application to view the numerical data. Figure 13 shows the format of the .csv data.

```
$ obd2csv -d obdgpslogger-vb.db
-o obdgpslogger-vb.csv
$ ls -lsa *.csv
112 -rw-rw-r-- 1 odroid odroid
111969 May 16 22:39 obdgpslogger-
vb.csv
```

Manage SQLite database

It is always useful to access the raw data collected by OBDGPSLogger, so that one can examine it for debugging purposes, or to learn about the schema in order to use the more powerful MySQL database system. Once the previously outlined installation process has completed, you can point your browser to http://<C1-IP-Address> to start the phpLiteAdmin application.

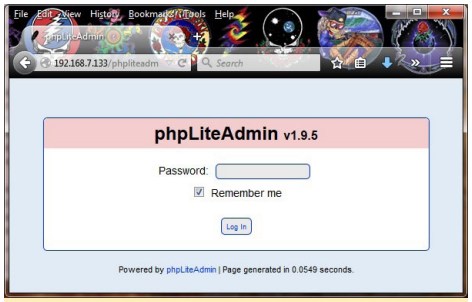


Figure 14 - phpLiteAdmin Login

Figure 13 - CSV data

obdgpslogger-vb.csv - OpenOffice Calc

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	obd.temp	obd.rpm	obd.vss	obd.maf	obd.throttlepos	obd.time	obd.trip	obd.ecu	(7.107*obd.vss/obd.maf) as mpg	gps.lon	gps.lat	gps.alt	trip.tripid
2	88	751.5	5	8.18	16.862745	315535124	2	0	4.344132	0	0	0	2
3	88	860.75	5	4.12	16.470589	315535126	2	0	8.625	-122.031818	37.32318	71.3	2
4	88	739	5	4.09	16.078432	315535127	2	0	8.688264	-122.031801	37.32319	70.9	2
5	88	696.75	3	3.29	16.078432	315535128	2	0	6.480547	-122.031789	37.323197	70.8	2
6	88	602.25	3	4.35	16.862745	315535130	2	0	4.901379	-122.031789	37.323214	71	2
7	88	744.5	3	4.32	16.862745	315535131	2	0	4.935416	-122.031789	37.323214	70.9	2
8	88	732	0	4.26	16.862745	315535132	2	0	0	-122.03179	37.32321	70.6	2
9	88	697.5	0	4.25	16.862745	315535134	2	0	0	-122.031793	37.3232	70.5	2
10	88	724	0	3.6	16.078432	315535135	2	0	0	-122.031796	37.323188	70.3	2
11	89	826.5	0	3.95	16.470589	315535136	2	0	0	-122.0318	37.323182	70.7	2
12	89	703	0	5.87	18.039215	315535138	2	0	0	-122.0318	37.32318	70.8	2
13	89	1045.25	0	7.82	19.215687	315535139	2	0	0	-122.031804	37.323184	70.8	2
14	89	1192.75	6	11.51	20.784313	315535140	2	0	3.704778	-122.031811	37.323183	70.7	2
15	89	1434.25	8	6.34	17.647058	315535141	2	0	8.967823	-122.031834	37.323189	70.9	2
16	88	1246	13	3.6	16.078432	315535143	2	0	25.664167	-122.031854	37.323202	71.1	2
17	88	848.25	12	3.54	18.039215	315535144	2	0	24.091526	-122.031885	37.323213	70.8	2
18	88	830.25	7	3.23	16.078432	315535145	2	0	15.402167	-122.031907	37.323227	70.7	2
19	88	749	6	3.32	16.078432	315535147	2	0	12.843976	-122.031938	37.323238	70	2
20	88	830.25	7	4.25	16.862745	315535148	2	0	11.705647	-122.031949	37.323254	69.8	2

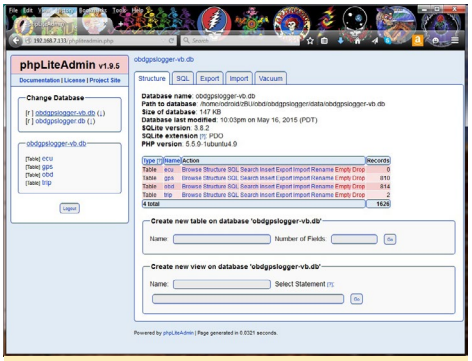


Figure 15 - phpLiteAdmin welcome

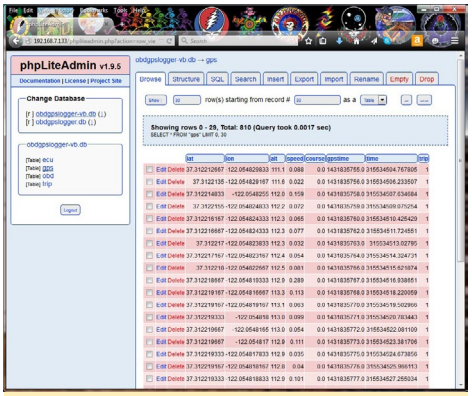


Figure 16 - GPS data

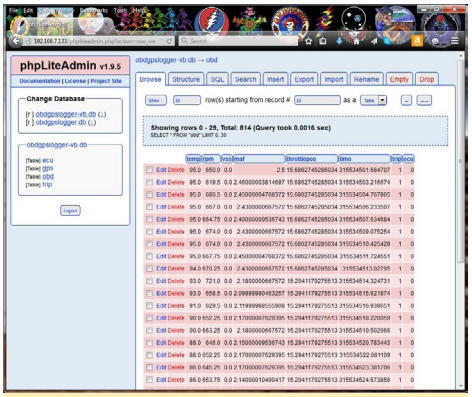


Figure 17 - OBD data

Use the password odroid, as configured previously, to log into the application. You will be then be presented with a welcome screen as shown in Figure 15. Select the tables from the left menu according to Figures 16 and 17. Now, you have everything needed to enhance the discussed open-source tools or build your own.

Disclaimer

As always, be cautious when manipulating your setup in a vehicle. Ensure

safety first before attempting any activity on the setup. HardKernel and the contributors of these articles may not be held liable for possible mishaps during your experiments.

Acknowledgements

Gary Briggs (chunky@icculus.org), the author of OBDGPSLogger, has kindly consented to the use of his software for publication in this article. Thanks to Gary on behalf of the ODROID community.

Additional Resources

- ODROID forum post: http://bit.ly/1Eu8HTB
- OBD GPS home page: http://bit.ly/1AuOe66
- ROOT data analysis framework: http://bit.ly/1Sztekz
- Introduction to Bluetooth programming: http://bit.ly/1AuOfqw
- PHPLiteAdmin: http://bit.ly/1dtZUw0
- Installing Google Earth for Linux: http://bit.ly/1FMzGz7
- PyOBD reference: http://bit.ly/1JSnbnt
- Bluetooth adapter: http://amzn.to/1FMzJeq
- OBDII adapter: http://amzn.to/1AuOh1V

ODROID FORUMS

THE PERFECT PLACE TO COMMUNICATE WITH HARDKERNEL DEVELOPERS

by Rob Roy

The ODROID forums have been the central meeting place for the growing Hardkernel community for several years, with over 11,000 members as of June 2015. You can discuss ODROIDs with Mauro, the lead Linux kernel developer, and Justin, the CEO of Hardkernel, along with a growing team of developers who donate their time to helping you get the most out of your ODROID. Check it out at http://forum.odroid.com!

LINUX GAMING: NINTENDO 64 EMULATION - PART 2

EMBARK ON THE ULTIMATE 90S GAMING JOURNEY

by Tobias Schaaf

Part 1 of this article introduced the latest version of the Nintendo 64 emulator for Linux and compared its performance on all of the current ODROID boards. This second part presents an overview of some of the more popular Nintendo 64 games, including Mario Kart, Mario Party, Paper Mario, Star Fox, Star Wars, Starcraft, Super Mario, Super Smash Bros, and Legend of Zelda.

Mario Kart 64

Mario Kart is very well-known racing game franchise from Nintendo, starring the most famous Nintendo characters like Mario, Luigi, Peach, Yoshi, Donkey Kong, Bowser and others. One of the big benefits of this game is that you can play it with up to 4 players at the same time.

I'm not really a fan of the series, especially the Nintendo 64 version, which is graphically poor in my opinion. Although the N64 is known for its 3D capabilities, Mario Kart 64 uses mostly 2D sprites, which don't look good. The only 3D elements of the game are the ground that you are driving on, and some obstacles and bridges, which makes the game very unattractive to play.

U3

When I first ran the game without frameskip, it was rather laggy. Since the game mostly uses old 2D sprites, it really made me wonder why this game needs so much CPU power. However, once I activated frame skipping, it worked fine on the U3. There is some small delay in the sound while using the menu, but nothing that's really troublesome. In-game racing works fine without lags or slowdowns, and multiplayer with several controllers is working perfectly as well.

C1

While the menu is slow, the in-game experience is good and seems to work at full speed using the Rice plugin. It's definitely playable, although you get a much better experience on the U3 or XU3 rather than the C1. When I re-tested it using glide64mk2, the game ran fine, although it had some glitches with the shadows and ground textures.

XU3

Mario Kart 64 had no issues on the XU3. It ran at full speed and could easily be controlled with an the XBox 360 controller.

ODROID MAGAZINE 17

Mario Party

Mario Party is a type of board game in which you play with or against up to 4 players in different kinds of mini-games. The game is quite fun, although sometimes I have a hard time figuring out the controls for certain mini-games. It's probably suited for all ages, from small children to adults as a party game, or just to have some fun.

U3

The U3 experience was flawless, and the game ran at full speed without any issues. I saw a flickering screen on the split screen of one mini-game once, but the moment the action started, it was gone and therefore fully playable.

CI - rice plugin

The menu was a little slow at first, and when I was actually on the map to select a game, I was rewarded with a very fluent movement, like in Mario Kart. However, when I tried to start a game I only saw a white screen. I heard everything running in the background, and clicking buttons triggered certain actions which I could hear, but I could not see anything besides a blank screen. When I tried again with a different game mode, I was able to see a few parts of the game, but major parts were missing, and the moment that I started a mini-game, I only got a black screen.

CI - glide64mk2

While the game didn't run using the rice plugin, it worked fine with glide64mk2, although it was a little slow. Most scenes in game are full speed, so I consider this game playable under glide64mk2 in 16-bit.

XU3

The XU3 had no issues at all playing this game. It ran smoothly, which was not surprising considering that it also ran well on the U3. Overall, the gaming experience was quite nice.

Paper Mario

Paper Mario is a mix between a jump and run game like Super Mario and an RPG game like Final Fantasy. It has nice graphics, and although the world is 3D, Mario himself is only 2D. He's actually a paper figure. The gameplay is very unique and is really fun to play. It's hard to describe, but you should definitely give this one a try!

U3

The U3 experience for Paper Mario is really good. The overall speed was very good, and I enjoyed the game a lot on the U3.

ODROID MAGAZINE 18

C1 - rice plugin

The experience on the C1 is hard to describe. At first, the game was not working at all. After a laggy introduction, the main menu did not show up. After 10 or 15 minutes, another type of introduction seemed to show up, which was basically just a scrolling background picture. Another 10 to 20 minutes later, the picture changed again and suddenly I saw the start menu. I created a new save state and started a new game. Again, I was presented with a single background picture. It seems the game is not working at all on the C1, or it might take hours for it to start. The C1 should be able to play the game in rather a decent speed, but unfortunately, the faulty drivers and graphics support prevent the system from working properly.

C1 - glide64mk2

This game works with glide64mk2 at full speed. Similar to the U3, it has glitches with the shadows and ground textures, but besides that, the game is running very well.

XU3

The libretro core did a very good job with this game. None of the U3 glitches with glide64mk2 could be seen. The shadows were perfect, speech bubbles were fine, and I could read what the stars were saying. The overall speed was perfect as well. I really like playing this game on the XU3.

Star Fox 64

Star Fox 64 is a remake of the Super Famicom/SNES game Star Fox, which was one of the first 3D space shooters. The N64 version was famous for its very good graphics and especially for its voice acting. The often funny lines of your comrades through the radio, the intense battles, and the good graphics make this game really fun to play.

U3

The game runs very well on the U3. It had some slow downs on the galaxy map where you select the mission, and the shadows are too dark. The lighting does not work correctly which means the game is very dark in some scenes. Besides that, the game works perfectly well at full speed.

C1 - rice plugin

The C1 does well with this game. The rice video plugin looks a lot better when rendering shadows than the glide64mk2 on the U3, so the scenes are not as dark. Besides that, the performance of the C1 is slower than on the U3, and the mission briefing is slightly laggy. While the U3 has a slowdown on the Galaxy map where you can select your mission, the C1 hangs very badly, but since it's just for selecting your mission it doesn't affect game play that much. When you're finally on the hunt and shooting through the game, the game runs at full speed without issues, and is actually nice to play on the C1.

ODROID MAGAZINE 19

C1 - glide64mk2

Similar to the U3, the gaming experience is rather good. It's about the same speed as on the U3 and has the same issues with the shadow, but besides that, the gaming experience is nice and only slows down on the galaxy map.

XU3

As usual, the XU3 experience is the best. The game runs smoothly, but slows down on the galaxy map. The graphics look great on the XU3, and the game runs very smoothly.

Star Wars Episode I – Racer

I played this game many years ago on the PC with my 3DFX Voodoo graphics card, which used the "glide" that's included in some of the graphic plugins for mupen64plus. The game is about the Pod Racer in Episode 1 of Star Wars. It's a very fast racing game with nice graphics and destroyable objects, and you can upgrade your pod to make it faster or easier to handle.

This game actually uses the memory expansion pak on the N64 which improved the graphics, and the rumble pak is also supported. However, the N64 version doesn't compare to the PC version in terms of graphics, and is also missing the multi-player mode, although it's still a nice racing game.

U3

The experience on the U3 is very good. The game runs fluently and quickly, and doesn't seem to have glitches. Some of the shadows are too dark, but that's something you only experience in the menu.

C1 - rice plugin

Once again, the C1 has issues with this game related to the rice video plugin, since the same issues happen on the U3 when the video plugin is switched to rice. The picture was distorted and cut off in some scenes. The game works perfectly fine using glide64mk2 at full speed with no issues.

XU3

The game works very well on the XU3. I finally figured out how to use the booster, and I also saw a two player option. It seems that if the game finds more than one controller connected, it offers a multiplayer option. The gaming experience was flawless and at full speed.

Star Wars: Rogue Squadron

This is named as one of the best N64 games ever made, where you fly an X-Wing to right against the evil Empire. I played the game on the PC when it came out, and it was quite fun. I was looking forward to trying it on the ODROID. I've read that this game requires the memory expansion pack in order to launch. However, no matter what I tried, I wasn't able to get this game to work on any platform or with any graphics plugin. Both the mupen64plus and libretro core emulators either crashed or stopped responding.

ODROID MAGAZINE 20

StarCraft 64

StarCraft is a very famous RTS game. It's one of the best strategy games ever made, and is still played in professional gaming tournaments. The Nintendo 64 game is a very good remake with reduced graphics, stripped videos, and minimal music. It's a nice strategy game, and I found it interesting that I was able to play it on a Nintendo 64 emulator.

U3

The game runs surprisingly well on the U3. There are some speed issues on the menu, but as soon as you are in the game, it works well, although the sound is a little bit delayed, especially in bigger battles. You can hear units die after they have already disappeared from the screen.

C1 - rice plugin

StarCraft 64 ran surprisingly well on the C1. It seems to work best using the rice video plugin. However, when using the glide64mk2 plugin, the menu is so slow that you can't select the mission that you want to play. Therefore, the game is not playable under glide64mk2. The in-game speed would probably be fine, but since I couldn't get past the menu, there is no way to tell.

XU3

I actually had a lot of issues getting StarCraft 64 to run on the XU3. The game was very laggy at first, and switching from glide64 to rice or gln64 exhibited strange issues. Rice and gln64 were really fast on the XU3 menu, and everything was full speed. But both rice and gln64 had major graphical problems, which made the game unplayable. After some investigation on the slowdown of glide64, I found out that reducing the rendering resolution increased the speed. The game is displayed in 1080p no matter which resolution you choose, but the resolution at which the characters and objects are rendered can be changed on the XU3. I found that using a resolution of 800x600 or below gave the best performance.

Super Mario 64

Super Mario 64 was the launch title for the N64, and what a launch title it was! This game boosted the N64 to the top of its class by showing what the console was capable of, and once again, made Mario the star of the Nintendo franchise.

U3

On the U3, Mario 64 has some glitches with shadows,

ODROID MAGAZINE 21

textures and lighting, but besides that, the game runs at full speed.

C1 - rice plugin

Mario 64 seems to be running a little below full speed on the C1, but it is still playable with the rice graphics plugin. The speed is slightly better with glide64mk2 than with the rice plugin, but it occasionally drops below full speed. It also has the same issues as the U3 glide64mk2 plugin with rendering ground textures and shadows.

XU3

The game is running fine on the XU3, with no issues or glitches.

Super Smash Bros

This game introduced a new genre of brawler games. It was a major success on the N64, and led to a lot of sequels. You can choose between famous Nintendo characters such as Mario, Yoshi, Princess Peach and many more, and fight against other characters.

U3

The gaming experience for Super Smash Bros on the U3 with mupen64plus and glide64mk2 plugin is very nice. Even the menu is working at a decent speed. There are some glitches with shadow and text, but nothing serious, and only the text issue is noticeable.

C1 - rice plugin

The game was too slow under rice to be playable. The menu, introduction, and gameplay were laggy. However, Super Smash Bros runs much better with the glide64mk2 plugin, and you can actually play it full speed, although it has the same glitches as the U3 version.

XU3

While in the menu, there is some lagging and slow downs, but the game runs perfectly fine otherwise. It was really fun to play.

ODROID MAGAZINE 22

The Legend of Zelda: Majora's Mask

I don't know much about the Legend of Zelda games on the N64, but I do know that this game involves having 72 hours to save the world, and you have different masks to help you in your cause. You can use the "Ocarina of Time" to travel back in time and start the 72 hours over and over again until you finished the game.

U3

Although the game speed is very good, the glide64mk2 plugin once again has issues with being too dark. Since it can't do the blurry effect, the game stays at full speed the entire time. However, because it's too dark, it's sometimes hard to find a way, but it's not as dark as it is when played on the XU3, where nothing is visible. I consider this fully playable.

C1 - rice plugin

The game worked surprisingly well on the ODROID-C1 with the rice plugin. There were no graphical issues, but the introduction and some scenes were slightly laggy. Overall, the game is very playable on C1 with the rice plugin.

C1 - glide64mk2

The game runs at nearly full speed, but suffers from the same darkness issue as the U3. Rice is probably the best plugin for use with this game when played on the ODROID-C1.

XU3

The overall experience of the game is quite good. When there are cutscenes with the blurring effect, the game slows down and becomes laggy. However, since only occurs in cutscenes, the gameplay is fine. However, there's another issue which is related to glide plugin, which is that the graphics are too dark, makes it hard to figure out which way to go. It got so dark that I switched to the gln64 plugin, which had some minor glitches with the ground, but otherwise worked perfectly at full speed. It was not so dark that you couldn't see where to go, so using gln64 as a plugin for this game worked great.

The Legend of Zelda: Ocarina of Time

This is the predecessor of Majora's Mask. I actually had a hard time enjoying the game, but I know that it is supposed to get better over time, and there must be a reason why so many have it on their top 10 list, so I gave it a try.

U3

Generally the game works fine and is at full speed, with some minor issues with shadows and ground textures. In some places, it

ODROID MAGAZINE 23

is too dark, but it is still fully playable.

CI - rice plugin

Similar to the other Legend of Zelda game, this one works very nicely on the ODROID-C1 using the rice plugin. With the glide64mk2 plugin, the game was not entirely full speed, and exhibits the typical ground texture and shadow issues.

XU3

The experience on the XU3 is superb. I didn't see any glitches or slow downs so far, although I didn't get very far in the game. It's a really nice experience.

High Resolution Textures

After trying out different games, I checked on what else could be done with the emulators, and I found out that there are some high resolution texture packs that offer much better graphics. I tried a few of them to see what they look like in order determine if they would work on the ODROIDs. Mupen64plus standalone emulator offers the possibility to use high resolution textures for N64 games which can improve gaming experience by giving a new look to the games, but this option is not available for other emulators.

To use the high resolution textures, download them from `http://bit.ly/1Jvpahr` and copy them to the directory `~/.local/share/mupen64plus/hires_texture/`. Some of the textures are complete rewrites of the game graphics. Make sure to place the

Super Mario 64 with standard textures and high resolution textures side by side.

ODROID MAGAZINE 24

textures in a folder with the "short name" of the game in capital letters. For example, Mario 64 is "SUPER MARIO 64", and Mario Kart 64 is "MARIOKART64".

A complete remake of the Mario 64 textures gives the game a modern look

Conclusion

Nintendo 64 emulation is generally working very well on ODROID devices, especially on the U3 and XU3. The C1 has a lot of issues which prevent it from offering the same gaming experience as on the other ODROID devices. The rice plugin, which works without having to change color depth settings on your image, has major issues with many games, but does a rather good job on other games.

The glide64mk2 plugin only works under 16-bit, and although most games are running nicely, the ones that do run better with the rice plugin require a reboot in order to be able to use it, since rice isn't working with 16-bit. This leaves me rather unsatisfied, since I always had to reboot the entire ODROID in order to switch between different graphics plugins on the C1. The U3 and XU3 can do this without rebooting the entire system, which makes it much easier to switch between the plugins.

Also, using 16-bit color depth prevents different applications such as XBMC from running properly, which causes you to choose a emulator frontend that actually supports 16-bit mode, or else you are forced to start N64 games through a Terminal window.

This all makes me believe that C1 is not really suitable for N64, at least under Linux. I think that the best way to play N64 games on the C1 is probably through the Android app or a highly modified version using fbdev drivers and some scripts that are able to switch color depths and applications to run. That setup would be very inconvenient and certainly not suitable for beginners.

The U3 and XU3 both measure up very well when it comes to N64 emulation. Being able to switch between graphic cores easily is a big benefit over the C1. N64 games seem to need some occasional tweaking, and if you look at the configuration options for either glide64mk2 or rice on the mupen64plus standalone emulator, there are a lot of options to choose from.

The XU3 is the only board that can use libretro core of mupen64plus with Retroarch at the moment. It integrates the controllers very nicely, and you can easily adapt your gamepad layout to your own needs and have various controllers supported. Also, the XU3 has extra CPU power, which often make the difference between full speed or "nearly" full speed. The U3 does a very good job in emulating N64 games, and being able to use high resolution textures in mupen64plus is really a cool thing to have.

ODROID MAGAZINE 25

3.2” TOUCHSCREEN DRIVER INSTALLATION

FOR THE ODROID-C1

by Bo Lechnowsky and Owen Browne

The 3.2” touchscreen for the ODROID-C1 is one of the more unique peripherals available from the Hardkernel store at http://bit.ly/1KYqWWw. The touchscreen comes with small stylus, and is able to display the Linux desktop in a sharp 320x240 resolution, perfect for use in embedded projects such as robotics, vehicle computers, and home automation. The following steps outline how to install the driver on any Ubuntu 14.04 distribution manually. To install the drivers automatically instead, please refer to the “Automatic installation” section at the end of the article.

Manual installation

First, install the dependencies and update the device operating system:

```
$ sudo apt-get update && sudo apt-get install fix-w1-
blacklist
$ sudo apt-get dist-upgrade
$ sudo apt-get upgrade
```

Then, run modprobe, and change rotate=0 if you want to run the display in portrait mode (vertical):

```
$ sudo modprobe spicc
$ sudo modprobe fbtft_device name=odroidc_tft32 ro-
tate=270 \
  gpios=reset:116,dc:115 speed=32000000 cs=0
```

To enable the X11 desktop, create a file called /usr/share/X11/xorg.conf.d/99-odroidc-tftlcd.conf with the following contents:

```
Section “Device”
   Identifier    “C1 fbdev”
   Driver         “fbdev”
   Option         “fbdev” “/dev/fb2”
EndSection
```

X Windows may then be started:

```
$ sudo startx
```

Disable Xorg by editing the file /etc/init/lightdm.override to contain the following, save the file, and reboot:

```
manual
```

After the device has rebooted, run con2fbmap:

```
$ con2fbmap 1 2
```

Next, add the following two lines to /etc/modules, making sure to change rotate=0 if you want to run the display in portrait mode (vertical):

```
spicc
fbtft_device name=odroidc_tft32 rotate=270
gpios=reset:116,dc:115 speed=32000000 cs=0
```

Add the following line to the end of the file /media/boot/boot.ini:

```
fbcon=map:22
```

Add this line to /etc/rc.local before the exit command at the end:

```
startx &
```

Reboot again and test your touchscreen by creating a file under /etc/udev/rules.d/95-ads7846.rules with the following contents on a single line:

```
SUBSYSTEM==”input”, ATTRS{name}==”ADS7846 Touch-
screen”, ENV{DEVNAME}==”*event*”,
SYMLINK+=”input/touchscreen”
```

Next, apply the new module by typing the following commands into a Terminal window:

```
$ sudo modprobe spicc
$ sudo modprobe -r ads7846
$ sudo modprobe ads7846
```

Then, check to see if a touchscreen node exists by inspecting the output of the following command:

```
$ ls /dev/input/touchscreen
```

Next, install the event and touchscreen libraries:

```
$ sudo apt-get install evtest tslib libts-bin
```

Run a test to see if the touchscreen responds to touch. The text may be exited by pressing Control-C:

```
$ sudo evtest /dev/input/touchscreen
```

The device needs to be calibrated in order to map the touch events to the screen properly. To do so, remove the previous calibration:

```
$ sudo rm /etc/X11/xorg.conf.d/99-calibration.conf
```

Then, create a directory in the X11 directory if it's not already present:

```
$ sudo mkdir /etc/X11/xorg.conf.d/
```

Depending on whether the touchscreen will be used in portrait or landscape mode, follow the steps in the relevant section below.

Calibration (landscape)

Add these lines into /etc/X11/xorg.conf.d/99-calibration.conf:

```
Section "InputClass"
        Identifier      "calibration"
        MatchProduct    "ADS7846 Touchscreen"
        Option  "Calibration"   "15 3836 4020 336"
EndSection
```

Calibration (portrait)

For portrait mode, the rotate=0 option should be enabled in the previous steps. Add these lines into /etc/X11/xorg.conf.d/99-calibration.conf:

```
Section "InputClass"
Identifier      "calibration"
MatchProduct    "ADS7846 Touchscreen"
Option "Calibration" "37 5543 108 2290"
Option "SwapAxes" ""
EndSection
```

After rebooting, the touchscreen should be enabled and calibrated properly for use with the stylus.

Custom calibration

To perform a custom calibration, the program xinput_calibrator may be used:

```
$ sudo apt-get install xinput-calibrator && xinput_
calibrator
```

Automatic installation

The touchscreen driver installation may also be done automatically using a pre-built script from Ameridroid. First, disconnect the C1 power and HDMI cables, then connect the touchscreen to the GPIO header. Boot the device and expand the root partition using the ODROID utility, then connect the C1 to the Internet.

Next, change to the root user by launching a Terminal window and typing the following. You will need to supply the root password, which is "odroid" on the official Hardkernel images:

```
$ sudo su
# wget http://respectech.com/odroid/c1-touch.sh
# chmod 755 c1-touch.sh
# ./c1-touch.sh
```

Press "a", then the Enter key to run all of the installation steps automatically. If you haven't updated your distribution recently, go make yourself a cup of tea because it takes a while to update the entire operating system. The C1 will reboot after the installation process has completed, and the touchscreen will automatically boot into the Linux desktop.

Make sure to connect the screen correctly to avoid getting shocked

LINUX RETROARCH NINTENDO 64 CORE

FOR THE ODROID-U3

by Daniel Mehrwald

Retroarch, which is a multi-system game emulator available for both Android and Linux, is always under development, and its open-source code can be improved by anyone willing to submit a pull request to GitHub. Recently, the Nintendo 64 emulator core called mupen64plus added several improvements to provide smoother game play and better audio. To try it out on Ubuntu 14 or 15, first install Retroarch by typing the following commands into a Terminal window:

```
$ sudo add-apt-repository ppa:libretro/stable
$ sudo apt-get update
$ sudo apt-get install retroarch retroarch-* libretro-*
```

Then, download compile the latest version of mupen64plus:

```
$ git clone https://github.com/libretro/mupen64plus-libretro.git
$ cd mupen64plus-libretro
$ wget -O patch.txt http://pastebin.com/raw.php?i=XWBBFH7d
$ patch -p0 < patch.txt
```

Verify that the "-marm" gcc switch is present in the Makefile file on lines 92 and 93 by editing it in your favorite text editor:

```
CFLAGS += -marm -mfloat-abi=hard -mfpu=neon
CXXFLAGS += -marm -mfloat-abi=hard -mfpu=neon
```

Next, compile the mupen64plus core:

```
$ make -j5 V=1 platform='odroid odroid-u'
```

Copy the resulting .so file to the default core directory for Retroarch:

```
$ sudo cp libretro-mupen64plus.so /usr/lib/libretro/
```

Finally, launch your Nintendo 64 game using Retroarch with the new core:

```
$ retroarch -L /usr/lib/libretro/mupen64plus_libretro.so \
  ~/path/to/your-game.n64
```

The optimal settings from within RetroArch are:

Settings->Driver Settings->Audio Driver = alsathread
Settings->Driver Settings->Audio Resample Driver = sinc
Options->Core Options->GFX Plugin = glide64

For questions or comments, please refer to the original post at http://bit.ly/1d8VR87.

COMMUNITY IMAGES

FOR THE ODROID-C1

by @robrodrigues

One of the best features of the ODROID community is the wide variety of images that have been uploaded by forum members. In addition to the official images that are published by Hardkernel, community images offer specialized features that may be useful for specific purposes, such as a media center, file server, or network attached storage. Here is a list of some of the most popular images that are available for the ODROID-C1 as of May, 2015:

Triple Boot
http://bit.ly/1EBmLvt

Ubuntu & Debian Minimal
http://bit.ly/1DLugEZ

KitKat Pocket Rocket
http://bit.ly/1e8hLso

GameStation Turbo
http://bit.ly/1JpaccI

OpenElec
http://bit.ly/1FlifFG

OpenMediaVault
http://bit.ly/1FgU0Ho

Tiny Core
http://bit.ly/1Jpah00

DietPi
http://bit.ly/1d9bXP8

Fedora
http://bit.ly/1IFc2XD

Minimal Debian Wheezy
http://bit.ly/1QT8x1l

Nas4Free
http://bit.ly/1d9aPLt

Arch Kali
http://bit.ly/1EcjsKf

Arch Official
http://bit.ly/1IFc6H6

Max2Play
http://bit.ly/1Ecjzpg

NetBsd
http://bit.ly/1e8iuK8

Debian 7.8
http://bit.ly/1d9b6xS

http://bit.ly/1B4PfwC

More Debian images
http://bit.ly/1B4Pgk4

To suggest more images to add to the sticky post on the ODROID forums, please refer to the original post at http://bit.ly/1d9blZQ.

THE ORIGINAL ODROID

WHERE IT ALL STARTED

Edited by Rob Roy

When Hardkernel first created the ODROID family of single-board computers, it was intended for Android developers to be able to prototype apps on an inexpensive, pre-rooted device to facilitate app development. It was also well-suited for games, and was one of the more innovative devices of its time for Android gaming.

The ODROID has since blossomed into a wide product line, from wearables to media centers to desktop replacements, with many add-on components suitable for hardware tinkering, robotics, home automation, Linux and Android programming, and much more. What follows is the original unedited advertisement for the first ODROID device from the late 2000s, which was a handheld device with a built-in screen, gyroscopic sensors and controller buttons that was eventually retired in November of 2009. Enjoy this blast from the past!

HARDKERNEL

ODROID is a developer's dream device. The fully assembled device comes with debug board, source code and schematics. You can also communicate with engineers worldwide through the ODROID developer community. ODROID is based on a Samsung S5PC100 833MHz ARM's Cortex-A8 with NEON multimedia accelerator.

Demo Videos

RockOn demo:

`https://www.youtube.com/watch?v=yM7N3JDnX4k`

SlideMe and SocialDROID:

`https://www.youtube.com/watch?v=-8S-8gCa2bo`

Android 720p Video HDMI Demo:

`https://www.youtube.com/watch?v=zEWrV8LuX04`

Space Megaforce:

`https://www.youtube.com/watch?v=oj5sKT_2-Dg`

Speed Forge 3D HDTV demo:

`https://www.youtube.com/watch?v=M5fKrScVtP8`

Android Games

The chip's 3D OpenGL ES capability enables high speed 3D games such as Speed Forge 3D. The ODROID is connected to a 42" HDTV via the standard mini HDMI cable. The ODROID platform is completely open. You can develop, publish and play numerous games.

Video Playback through HDTV

ODROID supports up to 720P resolution video files. You can connect the ODROID to a big screen HDTV through the built-in mini HDMI connector. Fast web browsing, social networking and emailing is easy to use through the WiFi network.

Music

Choose from various music applications available in the market. Flicking cover flows allow a more enjoyable experience.

To see the original advertisement, please visit the Hardkernel archives at `http://bit.ly/1Gx5Lr1`.

COMPARISON OF THE TOP 4 SBCS

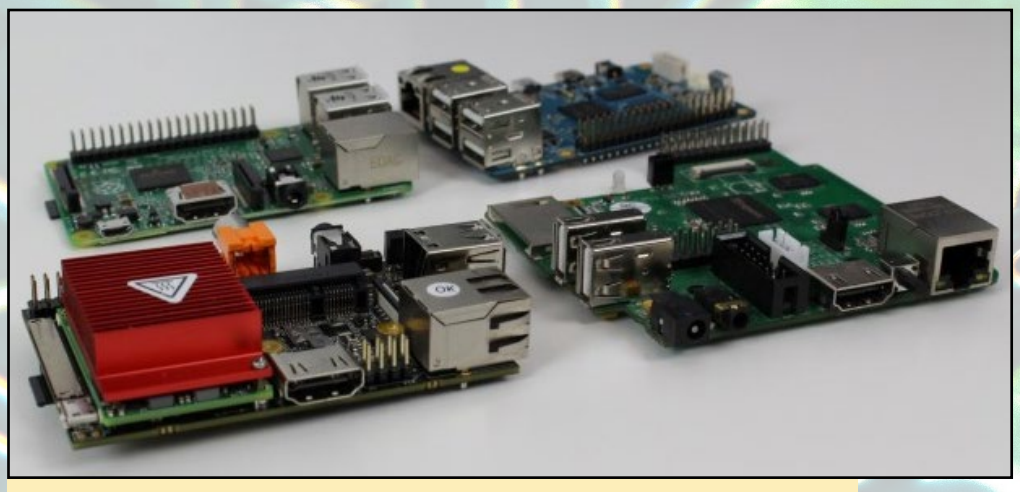
ONE BOARD TO RULE THEM ALL

by Gary Sims

Reprinted with permission from Android Authority (www.androidauthority.com)

The 2012 release of the original Raspberry Pi created a whole movement of hobbyists, developers, and educationalists, who used the ARM based platform to create, hack, and teach. Although the Raspberry Pi wasn't the first Single Board Computer (SBC) on the market, it succeeded for three important reasons. First, it was a full computer on a little board, it had a desktop and you could write computer programs on it; second, it had a set of user programmable GPIO pins, similar to those found on microcontroller platforms like the Arduino; third, and probably most importantly, it only cost $35.

Since then, the SBC market has grown significantly and the Raspberry Pi is no longer the only choice. Among the popular devices available are the ODROID-C1, the HummingBoard, the MIP Creator CI20, and the Raspberry Pi 2. Of course, the list of available boards is much longer, but these are the ones that I have personally tested.



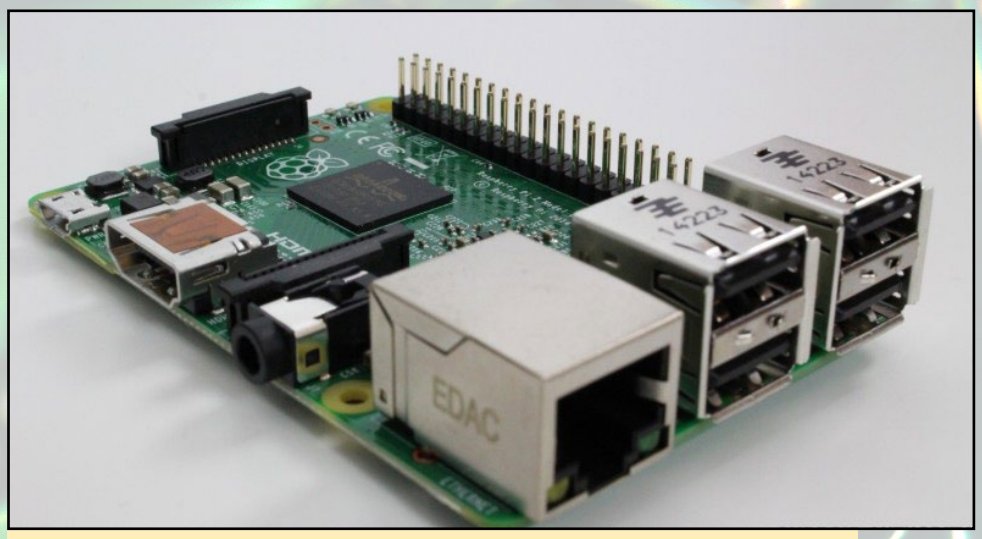
Comparison in size of all of the single board computers tested

The SBC market is heavily dominated by ARM and three of the four boards that we will be looking at use ARM based processors. The exception is the Creator CI20 which uses a MIPS processor. So before we compare the boards, let me formally introduce you to each one.

Raspberry Pi 2

Although the Raspberry Pi 1 was enormously successful, there was one complaint: the overall performance of the board was lacking, especially when running desktop applications. The performance was less than desirable because it used a single core CPU clock at just 900 MHz. Considering the cost, the innovative nature of the board, and its versatility, then the performance is perfectly understandable, but there was room for improvement. That improvement came in

the form of the Raspberry Pi 2, which uses a quad-core processor and doubles the amount of RAM. Even though the Pi 2 is more powerful and has more memory, the Raspberry Pi foundation managed to keep the price exactly the same, which is a guaranteed recipe for success.

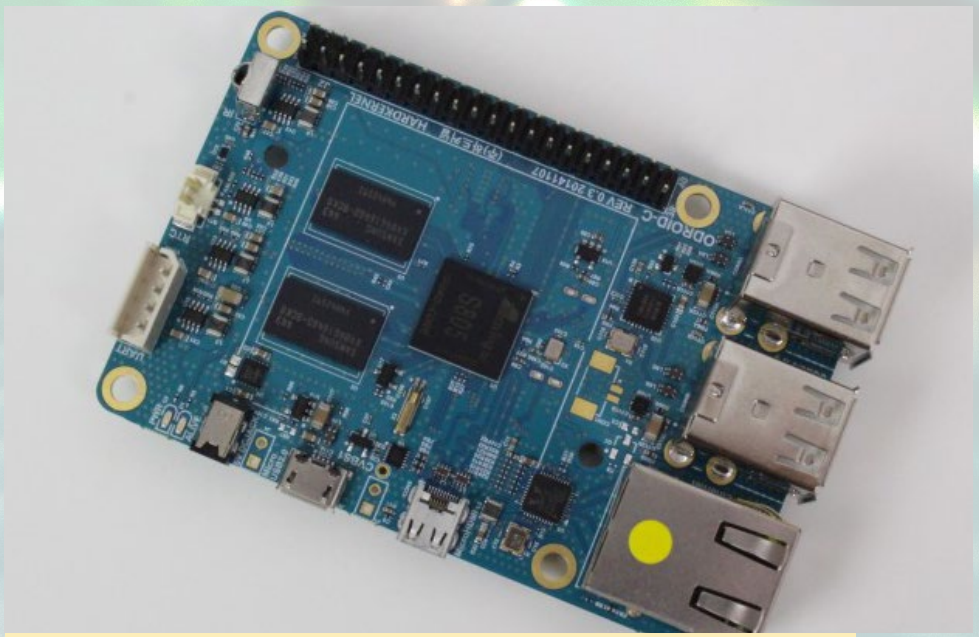


Raspberry Pi 2

ODROID-C1

One of the key reasons for the success of the Raspberry Pi was its price. While there are lots of other companies that make SBCs, there aren't that many who seem to be able to match the Pi's price point. Of course, some of the boards are only slightly more expensive than the Pi, and to be fair they often offer more functionality, as we will see with the MIPS Creator CI20.

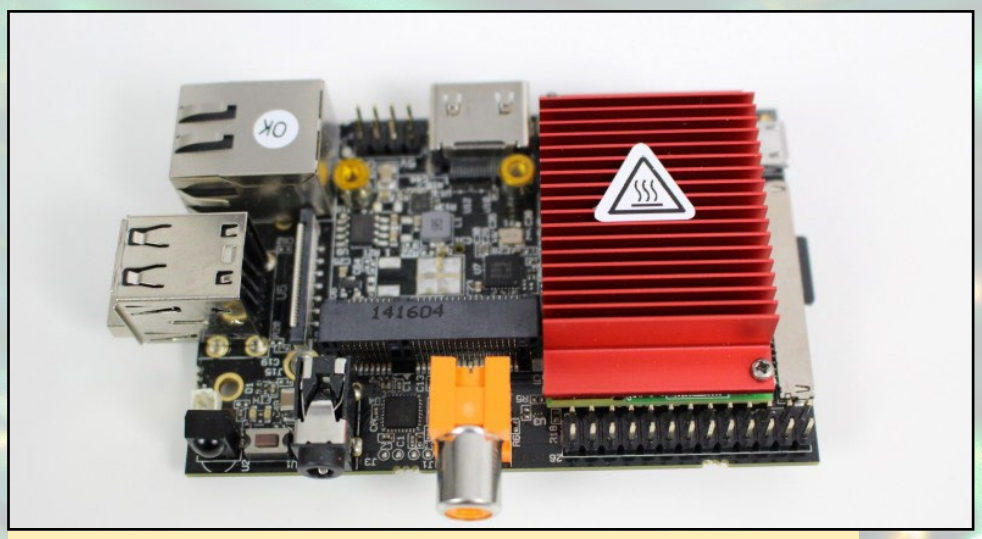
However one company that has managed to build a board for the same basic price as the Raspberry Pi is HardKernel. Called the ODROID-C1, it also costs $35. And like the Pi 2, it also uses a quad-core processor and comes with 1GB of RAM. The ODROID-C1 isn't the only SBC that HardKernel make, but it is the least expensive one.



ODROID-C1

HummingBoard i2eX

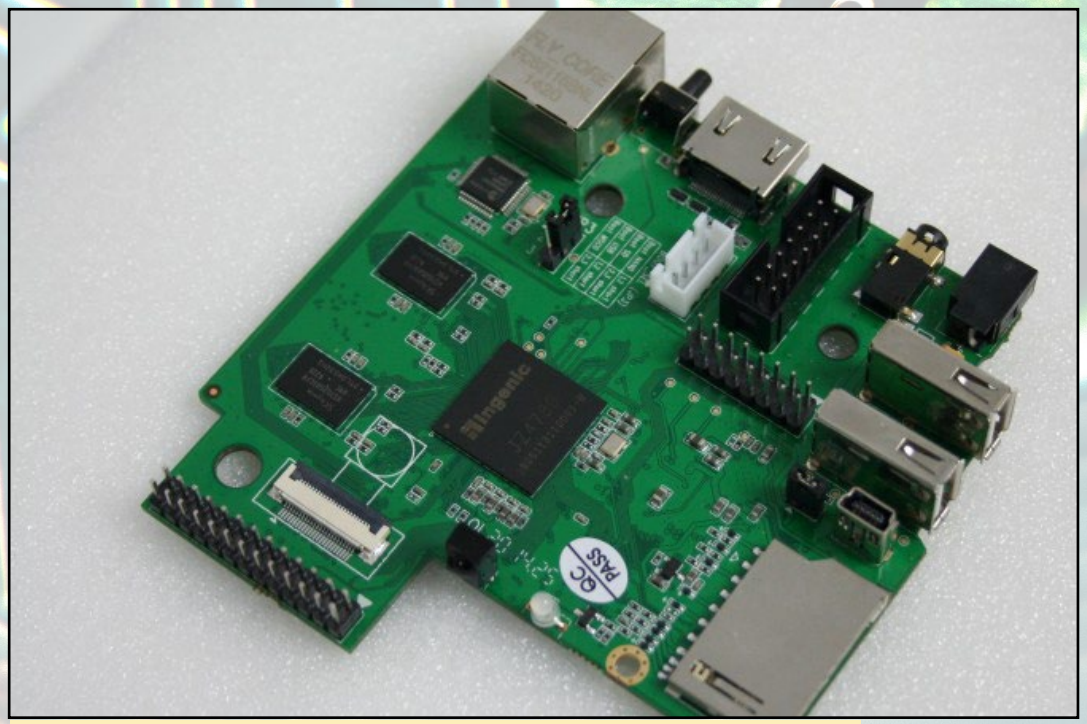
Another company which offers several different SBCs is SolidRun. All of their boards are built around Freescale's i.MX 6 series of processors. The i.MX 6 range is based on ARM's Cortex-A9 design, and scales from single- to quad-core. The HummingBoard i2eX uses a dual-core i.MX 6 processor, comes with 1GB of RAM, and has the same form factor as a Raspberry Pi 1 – it will even fit into a case designed for the first generation Pi.



Hummingboard

MIPS Creator CI20

The one board in our line-up which doesn't use an ARM based processor is the MIPS CI20 Creator. At its heart is a dual-core MIPS based processor coupled with a PowerVR GPU and backed by 1GB of RAM. It is also unique in that it includes its own built-in storage, plus Wi-Fi and Bluetooth. At just $65, it is more expensive than the ODROID-C1 or the Raspberry Pi 2, but you are getting more for your money.



Imagination CI20

Feature comparison

Now that you have been introduced to our four boards, how do they compare on paper? Here is a list of the specifications of each board:

Each of the boards in our test can run at least two operating systems, all of them run Linux, and most of them run Android. The one board which doesn't run Android is the Raspberry Pi, 1 or 2. The Raspberry Pi Foundation doesn't see Android as a priority, and there appears to be some porting difficulties due to some missing drivers from Broadcom. Of course, this could all change.

Device	ODROID-C1	Raspberry Pi 2	HummingBoard i2eX	Creator CI20
CPU	1.5Ghz quad core ARM Cortex-A5 CPU from Amlogic	900MHz quad-core ARM Cortex-A7 CPU from Broadcom	1GHz i.MX6 dual-core Cortex-A9 CPU	1.2GHz dual-core Imagination MIPS32 CPU
GPU	Mali-450 MP2 GPU	Videocore IV	GC2000	PowerVR SGX540
Memory	1GB	1GB	1GB	1GB
Storage	SD card slot or eMMC module	SD card slot	SD card slot	8GB onboard flash, SD card slot
Connectivity	4 x USB, microHDMI, Gigabit Ethernet, infra red remote control receiver	4 x USB, HDMI, Ethernet, 3.5mm audio jack	2 x USB, HDMI, Ethernet, 3.5mm audio jack, infra red remote control receiver	Ethernet, 802.11 b/g/n Wi-Fi, Bluetooth 4.0, 2 x USB, HDMI, 3.5mm audio jack
OS	Android, Linux	Linux, Windows 10	Linux, Android	Linux, Android
Connectors	GPIO, SPI, I2C, RTC (Real Time Clock) backup battery connector	Camera interface (CSI), GPIO, SPI, I2C, JTAG	Camera interface (CSI-2), GPIO, UART, SPI, I2C, PCI-Express Gen 2, mSATA II, RTC with backup battery	Camera interface (ITU645 controller), 14-pin ETAG connector, 2 x UART, GPIO, SPI, I2C, ADC
Price	$35	$35/£24	$110	$65/£50

Android does, however, run on the ODROID-C1, the HummingBoard and the MIP CI20 Creator. Currently all three only support Android 4.4 KitKat, but each one has the potential to run Android 5.0 Lollipop, however none of the board makers have officially released a ROM at this time. (Editor's Note: The ODROID-C1 now has a beta community image available for Android 5.0 Lollipop at http://bit.ly/1B5Ysqh).

To judge how well Android is supported on each of the boards I will use the following criteria: features, performance and support for Google's services. The two main Android features that distinguish one board from another are support for sound over HDMI and support for USB flash drives. The best board in terms of these features is the ODROID-C1. The HummingBoard and the CI20 don't support USB flash drives under Android, and the CI20 doesn't support sound over HDMI. Scoring each board out of 4 for features: the ODROID-C1 gets 4, the HummingBoard gets 3, and the CI20 scores 2.

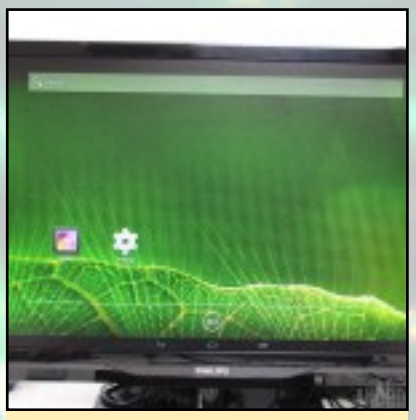
The next comparison is performance. Using AnTuTu as a guide to the relative performance, the ODROID-C1 scored 15887, and the HummingBoard-i2eX scored 12198. I wasn't able to test the CI20, but according to comments I have seen on the Internet, it scores less than the other two. So, scoring each board out of 4 for performance, the ODROID-C1 gets 4, the HummingBoard gets 3, and the CI20 scores 2.

Finally, in terms of support for Google Play and Google's services: the HummingBoard comes with Google Play pre-installed, whereas the ODROID-C1

doesn't include Google's services by default, but you can install them via an Android app supplied by Hardkernel. The CI20 doesn't include support for Google's service at all. Therefore, scoring each board out of 4 for Google Play support: the HummingBoard gets 4, the ODROID-C1 gets 3, and the CI20 scores 2.

ODROID-CI running Android

Hummingboard running Android



MIPS CI20 running Android

Since the Raspberry Pi doesn't support Android, it will score 0 for this section. The totals for this section are:

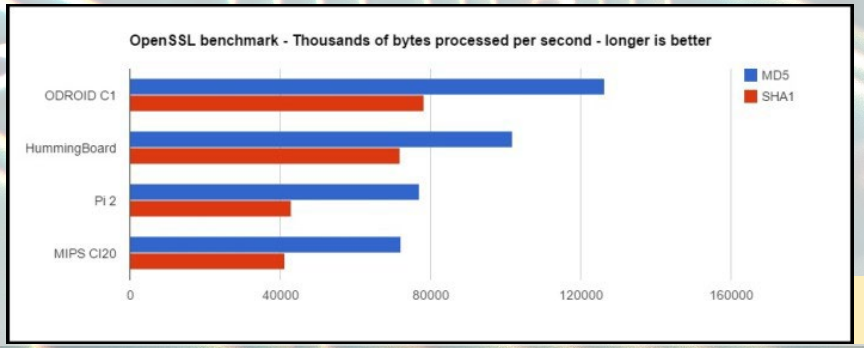
ODROID-CI	**11**
HummingBoard i2eX	**10**
CI20 Creator	**6**
Raspberry Pi	**0**

Linux

All four boards support Linux, and they all support it well. To try and judge which board supports Linux the best, I used the following criteria: the number of distributions supported, performance, and the amount of free memory available after a fresh boot into the desktop.

The board which supports the most Linux distros is the Raspberry Pi 2. Largely due to the sheer size of its user community, the Raspberry Pi is a popular platform and therefore receives the most attention in terms of porting. The scores for distro support is therefore: Raspberry Pi – 4, ODROID-C1 and HummingBoard – tied on 3, and CI20 – 1.

As for performance, the OpenSSL command line tool has a speed option which tests the performance of its various cryptographic algorithms. It also provides a good way to judge the relative performance of one CPU compared to another, as shown in Figure 7.



SBC OpenSSL benchmark

The SSL benchmark scores were quite revealing. The fastest board of the four, in terms of CPU performance without help from the GPU, is the ODROID-C1. Next comes the HummingBoard, followed by the Raspberry Pi 2. Last place, but not by much, goes to the CI20. As a result, the scores for performance are: ODROID-C1 – 4, HummingBoard – 3, Raspberry Pi 2 – 2, and the CI20 – 1.

Since these boards all have 1GB of RAM, it is important how much free memory remains once the board has booted to the desktop. The graphical user interfaces can be memory hogs and each of the boards uses a lightweight window manager to try and conserve memory. The results are for the default or recommended distro that can boot into the desktop without any additional installation and configuration by the user.

The most frugal board is the Raspberry Pi 2, which had 816360K free after booting. Next comes the CI20, which had 737436K free. The ODROID-C1 had 425836K free, and finally the HummingBoard had 313860K free. So, the scores for the free memory test are: – Raspberry Pi 2 – 4, the CI20 – 3, ODROID-C1 – 2, and HummingBoard – 1.

Collating all the score for this section, the results of the Linux tests are as follows:

Raspberry Pi	**10**
ODROID-C1	**9**
HummingBoard i2eX	**7**
CI20 Creator	**5**

ODROID-C1 running Linux

Raspberry Pi 2 running Linux

Kodi/XBMC

All four boards should support Kodi/XBMC. To test the performance of Kodi I used its internal codec information display to show the frame rate and the amount of CPU time being used to decode the video. I then produced a Full HD, 50Mbps version of my ZTE Blade S6 Plus review video and played it on each board.

The ODROID-C1 and the HummingBoard i2eX both did an excellent job and managed consistently to show the video at its full frame rate, while neither taxed the CPU too much in doing so. The same can't be said for the Raspberry Pi, which disappointingly could only manage 9 fps, instead of the needed 23.97 fps. Unfortunately I couldn't find an easily accessible version of Kodi to run on the CI20, and neither could I find a video player in the online repositories.

According to The Raspberry Pi Foundation the way Kodi works on the Pi is it

bypasses the GUI rendering, which means the frame rate reported by the codec overlay won't be accurate. As for the mouse lag, this is a known phenomenon and the best results will be achieved when using the keyboard or some form of remote control. The scores for this section are: ODROID-C1 – 4, and HummingBoard – 4, Raspberry Pi 2 – 2, CI20 – 0.

ODROID-CI running Kodi

Raspberry Pi running Kodi

Other operating systems

The big news that accompanied the launch of the Raspberry Pi 2 was that Microsoft will release a free version of Windows 10 for the Pi 2, which is aimed at creating Internet of Things (IoT) devices. Although the idea of Windows 10 running on a Raspberry Pi sounds intriguing, you might yet be disappointed, the IoT version of Windows could be quite limited, and in fact it may not even offer a desktop. Besides Windows 10, the Raspberry Pi 2 has support for RISC OS, NetBSD, FreeBSD, and OpenWrt.

As for the other three boards, they each have a measure of support for different OSes. For example, FreeBSD is known to run on the HummingBoard, while NetBSD has been ported to the ODROID-C1 and the MIPS CI20 Creator. There is also a work in progress to support OpenWrt on the CI20.

In a nutshell, the Raspberry Pi 2 has the widest OS support and the other three are very similar in the level of support offered. Therefore to score this section I will give the Raspberry Pi 2, 4 points. And the other three, 2 points each.

Community support

A big factor in picking an SBC is the size of the various online communities. How many people are there blogging about this board? Making videos about it? Writing books about it? Offering help in forums? There is little doubt that the Raspberry Pi community is the largest. This is mainly because of the success of the original Raspberry Pi, however it is already clear that the community has embraced the new Pi 2 board with the same passion. It is hard to judge between the online communities of the ODROID and the HummingBoard, but roughly speaking, in broad terms, they are approximately the same! The CI20 has the smallest of the communities partly due to its relative newness.

As a result, the Raspberry Pi 2 scores – 4, the ODROID-C1 and the HummingBoard – 3 each, and the CI20 – 1.

So which board is the winner?

The total scores are:

Device	ODROID-C1	HummingBoard i2eX	Raspberry Pi 2	MIPS Creator CI20
Android tests	11	10	0	6
Linux tests	9	7	10	5
Other OSes, Kodi/XBMC, community size	9	9	10	3
Totals	29	26	20	14

The final results show that the ODROID-C1 the winner of our board showdown. This is perhaps a surprise, as you may have expected the Raspberry Pi 2 to win. The reason it scored so badly was its lack of Android support. But other than its lack of support for Android, the Pi 2 does have other weaknesses. It is easily beaten by the ODROID-C1 and the HummingBoard in terms of performance, and even the dual-core MIPS processor comes close to the Pi's performance level. Also, the current version of Kodi for the Raspberry Pi doesn't handle video that well, which might be fixed in the future, but at the moment the ODROID-C1 and the HummingBoard do a better job. However, if you need Android support then the ODROID-C1 is the clear winner.

For more information, or to post questions or comments, please visit the original article at http://bit.ly/1JoW8zT.

TEXT TO SPEECH
WITH THE ODROID-C1 USB AUDIO ADAPTER

By Bo Lechnowsky and Brad Wilson

We've had a few opportunities to learn new tricks while working on our OWEN robot (Odroid Walking Educational uNit). One of these hurdles was getting the robot to speak through the C1 running Ubuntu using lightweight software. The first step is connecting the C1 USB Audio Adapter, available at http://bit.ly/1RW7TS4, and configuring a Text To Speech (TTS) engine to use it.

1. Plug in the USB Audio Adapter

2. Install Festival TTS:

```
$ sudo apt-get update
$ sudo apt-get install festival
```

3. Set up ALSA to use the second audio output, which corresponds to the USB Audio Adapter:

```
$ sudo pico ~.asoundrc

pcm.  !default {
      type hw
      card 1
}
ctl.  !default {
      type hw
      card 1
}
```

4. Configure Festival TTS to use ALSA and 16-bit audio. To do so for the current user, type the following command:

```
$ sudo pico ~/.festivalrc
```

Alternatively, to have ALSA as the default for all users, type this:

```
$ sudo pico /etc/festival.scm
```

Add the following lines to the file:

```
(Parameter.set 'Audio_Command
"aplay -D plug:dmix -q -c 1 -t
raw -f s16 -r $SR $FILE")
(Parameter.set 'Audio_Method 'Au-
dio_Command)
(Parameter.set 'Audio_Required_
Format 'snd)
(Parameter.set 'Audio_Method
'linux16audio)
```

Save the file, then reboot:

```
$ sudo reboot
```

Now you can use Festival TTS with the C1's USB Audio Adapter! Here's a couple of example commands:

Example 1: A beautiful day message (echo text)

```
$ echo "It's such a beautiful
day! Why are you in front of the
computer?" | festival --tts
```

Example 2: What's today's date? (program output)

```
$ date '+%A, %B %e, %Y' | festi-
val --tts
```

ALL-IN-ONE DESKTOP CASE FOR ODROID-C1

by Dongjin Kim

In the movie Jobs (2013), Steve Jobs delivered 50 units of the Apple I computer board to a computer store called the "Byte Shop". The owner of the store, Paul Terrell, complained about it not having peripherals such as a case, display and keyboard. After some argument, Jobs said "all-in-one" to his colleagues when he walked out of the shop. My favorite part of this movie is the scene where Jobs says "insanely great!" when he first plays with the original Macintosh, which was a true all-in-one computer.

When I first saw a Macintosh Plus in a printing office near my home, I was about 16 years old, and loved its shape. I only realized it was a Macintosh after I began studying computer programming. Recently, I tried to buy one, and looked around the domestic secondhand market and eBay. There were only dirty or cracked machines available, and I considered buying one and cleaning it up myself. Eventually, I decided to make an imitation case at a 1:1 scale of the original Macintosh Plus using an ODROID-C1.

I couldn't find an official CAD drawing of the Macintosh Plus, and instead I found some projects which placed an iPad tablet or Micro ATX board into an original Macintosh case. I got the dimensions from one of those projects, and tried to duplicate the size and shape. The most important thing for me in designing the case was the display and floppy disk openings, since these are the signature elements of the Macintosh. The display had to be 9" since original Macintosh uses the same size of CRT, but it was not easy to find that size display, so instead I picked up a 9.7" IPS LCD panel.

I am a software engineer with basic hardware knowledge and no experience in using CAD for mechanical design, so I planned to create one with a traditional ruler and a pen, using a thick 3mm paperboard. However, paperboard was too weak for the actual size of a Macintosh Plus, and I doubted that it could hold the whole weight of peripherals like an LCD and hard drive, so I switched to 5mm acrylic, which is more sturdy.

Since I chose acrylic for the case, I couldn't cut all pieces of the design by hand, and decided to order the acrylic laser cutting based on a CAD drawing. Eventually, this project led me to learn how to use the SketchUp CAD tool, which is amazing. I finished the design more quickly than I expected, with the only problem being that I couldn't be sure if the acrylic pieces could be processed and assembled correctly per my design.

After a week, the 20 pieces of acrylic for the case arrived, and every piece seemed to be exactly manufactured as I designed. Immediately, I started to assemble all of the pieces, and fortunately there were no missing parts. However, I had added too much space at the connecting edges, so the pieces could be assembled but needed to be held tightly. This was expected, and I had already prepared the connectors with aluminum reinforcement.

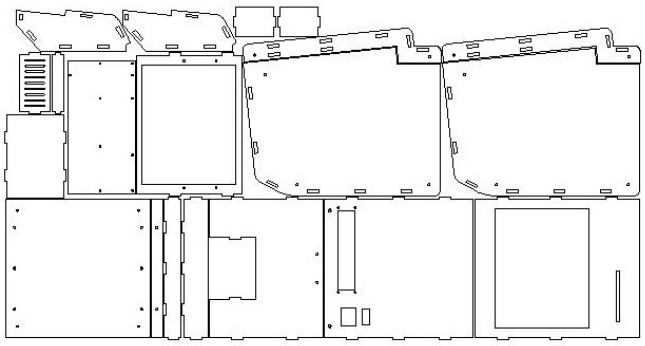


Figure 1 - Acrylic pieces as designed in the CAD software

My son wanted to help me out with completing the case, and asked if he could make an LCD part. He did well until he lost interest after a few hours and went out to play with his

Figure 2 - Assembled acrylic case

friend in the playground, so I finished assembling the case myself.

Once I connected my keyboard, mouse and speakers, the case looked amazingly like a real desktop PC. I planned to use a USB audio dongle with external speakers, but I decided

Figure 3 - Son assembling acrylic case

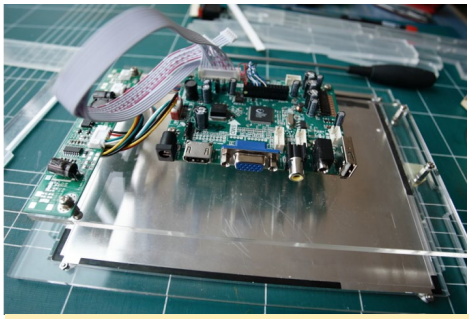


Figure 4 - ODROID-CI mounted inside acrylic case

Figure 5 - LCD panel mounted inside front of case

Figure 6 - Fully assembled case with keyboard, mouse and speakers

to add internal stereo speaker units instead, which was not in my original design. Therefore, I had to make a big hole for the 3" speaker units on the left and right side panels by hand. It took a few hours, and made my work table very dirty. Once I re-assembled the case with the internal speakers and started to play an Angry Birds game, I realized that this was the right choice.

I then discovered a critical problem after fully assembling the case. When it was completely closed, the USB devices couldn't be inserted, since the

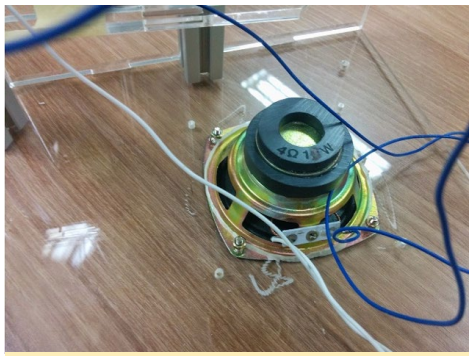


Figure 7 - Internal speaker mounted on front of case

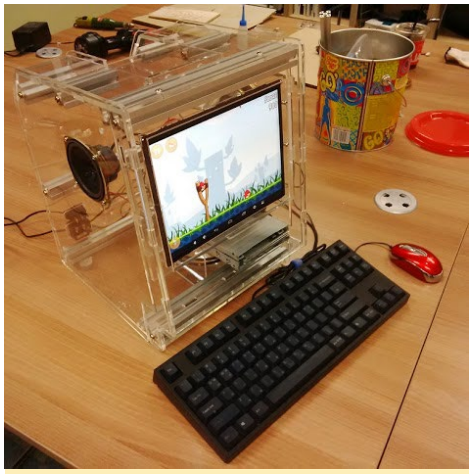


Figure 8 - First trial of assembled computer running Angry Birds

ODROID-C1 itself is in the middle of case. Therefore, I had to add an I/O expansion panel on the rear side, which I had missed while designing the case. Two UART connections and a couple of USB ports and RJ45 connectors seemed to be enough, and I made a panel for these connections, then had to do some solder work in order to connect the wires to the ODROID-C1's I/O. I also had to add an extension board on top of the C1 in order to accommodate the RS-232 signals for two UARTs for debugging and GPIO purposes. Meanwhile, the dual channel power supply arrived and was able to be mounted inside the case, which supplies 5V for the ODROID-C1 and 12V for the LCD and display board.

The LCD panel and display board have their own keypad and IR receivers, which was another problem since the

Figure 9 - Peripheral attachment area

keypad needed to be accessed easily and could not be mounted inside the case. So, I added more GPIOs for this keypad

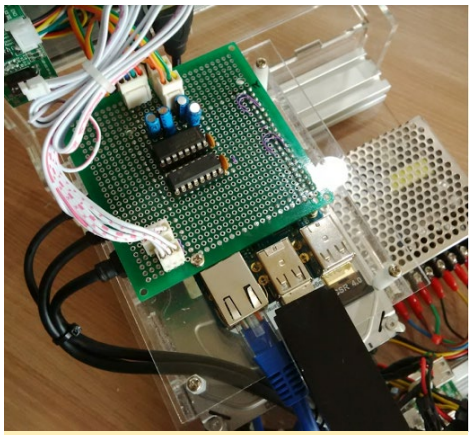


Figure 10 - Extension board

ODROID Magazine is now on Reddit!

ODROID Talk Subreddit
http://www.reddit.com/r/odroid

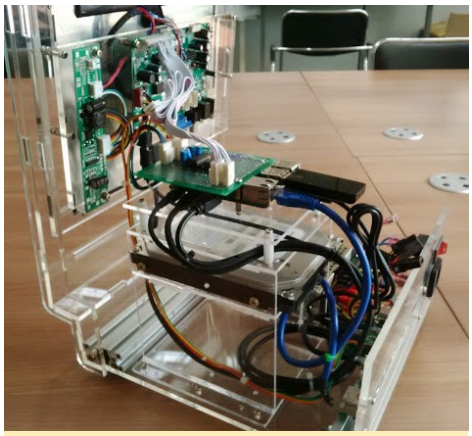


Figure 11 - Side view of partially assembled case

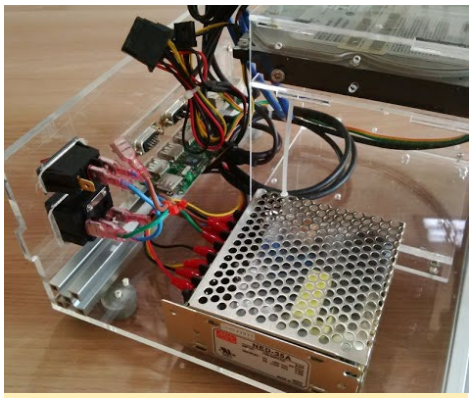


Figure 12 - Closeup of power supply

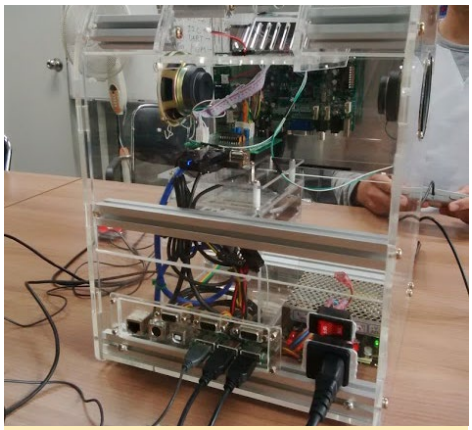


Figure 13 - Rear view

Figure 14 - Front view

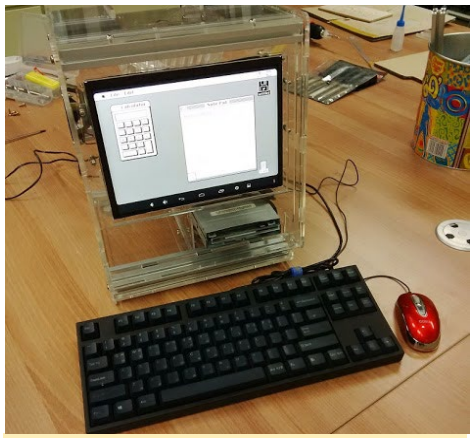


Figure 15 - Running Mini vMac, a Mac Plus emulator program

on an extension board in order to control the on-screen display using the WiringPi library running on the ODROID-C1, instead of relying on the display board's IR receiver.

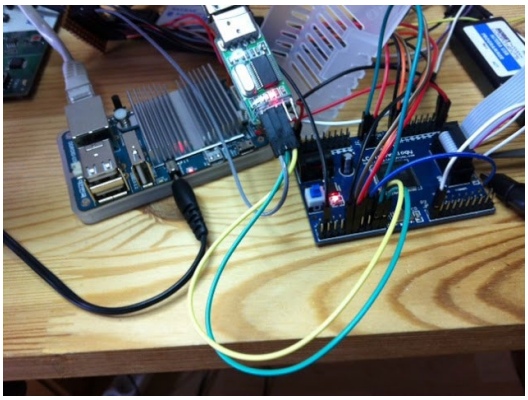
My computer actually looks like a transparent, futuristic Macintosh Plus when the Mini vMac software is running. I plan to further enhance the functionality, like supporting an SD card reader by disguising it as a 3.5" floppy drive connected to the ODROID-C1's socket. The important function for the 3.5" drive is to auto-eject like original Macintoshes do, so that the SD card can be ejected when a software command is sent. The case itself is almost 4kg when fully assembled, so I'd like to update the design to be lighter with more air holes, as well as make it easier to assemble.

For more pictures, visit the original desktop case post on the ODROID forums at http://bit.ly/1HFbmAE.

Now you can play Angry Birds on your own vintage Macintosh Plus!

GUZUNTY PI FOR THE ODROID-U3

USING A CPLD AS A PROGRAMMABLE LEVEL SHIFTER



by Carsten Foss

I have an ODROID-U3+ and wanted be able to connect to the serial console. The processor on the U3 is a Exynos 4412, which uses 1.8V for the I/O lines. The 1.8V I/O lines presented a challenge, as most of the inexpensive USB to serial adapters on eBay were either 5V or 3.3V, and not compatible with a 1.8V connector.

I looked into a very nice level-converter from Texas Instruments, the TXB0104, but then I remembered that I had a cheap Altera EPM240 CPLD (Complex Programmable Logic Device) board lying around. Having taken an online beginners VHDL (VHSIC Hardware Description Language) course this summer from PyroElectro (http://bit.ly/1PsRllZ), I thought about trying to combine the level-shifting, while also improving my VHDL skills.

As I had just seen the Hardkernel post about the new SPI (Serial Peripheral Interface) connector on the U3+ boards, I might even be able to do some SPI port expansion. Since there are 192 Macrocells in the EPM240 CPLD, it is a very capable chip: http://bit.ly/1EdwxD4.

Research

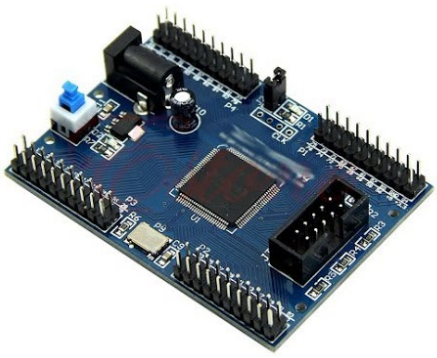
I had a look on the Internet for some interesting projects that I could use as a base, and found the Guzunty Pi project at http://bit.ly/1QVEdTN, which was just what I was looking for. It's an SPI IO expansion CPLD for the Raspberry Pi with a GPL licensed codebase. There is also a nice set of cores written in VHDL, available at http://bit.ly/1cJDZ30.

The heart of the Guzunty is a Xilinx XC9600XL CPLD. The designer chose the XC9500XL CPLD series due to its 5V tolerant pins on the CPLD. We cannot use the XC9500XL CPLD series for the U3, as it is not 1.8V compatible, so we need a CPLD that can handle 1.8V on the U3 input side. Besides that, I had already set my mind on using the USD$7 Altera MAXII EPM240 board that I already had in the drawer. Please note that the Guzunty Pi is not my creation, and I only ported the VHDL to the EPM240 board.

For programming the Altera CPLD, I used a cheap USD$6 Altera USB Blaster clone. Please note that there is also a red-colored EPM240 board available, which is not recommended for two reasons:

1. The VCCio1/2 traces are under the CPLD, making it impossible to modify the PCB. You have to lift the 3 x VCCio1 pins in order to make Bank1 operate at 1.8V, with a pin spacing of 0.5mm. The red version has the 3.3V Oscillator on Bank2, which is why it's recommended to use Bank1 at 1.8V.
2. It practically lacks every decoupling capacitor, that the EPM240 design notes specifies.

Figure 1 – EPM240 board



Blue CPLD board

I uploaded the schematic for the Blue Altera EPM240 board to the ODROID-U3 forums at http://bit.ly/1QVEXIw. The board comes with the following connectors:

U1: Altera MAXII EPM-240 CPLD

U2: 3.3V regulator

J1: DC power plug for connecting 5V

J6: Power switch which toggles the 5V to the regulator

P9: 50MHz oscillator, connected to GCLK0 (pin 12)

J2: Jumper to connect the on-board LED to CPLD pin 77

JTAG: 10-pin JTAG connector

CLK: 4-pin header for full access to the 3 remaining global clock pins

P1-P4: Almost all other pins routed to pin headers.

CPLD dedicated pins

The EPM240 has two I/O banks making it suitable for a 1.8V side, and a 3.3V side. Most pins on a CPLD are user-definable, but there are a few pins that are fixed, which are detailed in the list below:

1: VCC (power to the CPLD core)
2: VCCio (power to the I/O banks)
3: GND (ground)
4: Global Clocks (GCLK) (the EPM240 has 4 of these, two on each I/O bank)
5: JTAG pins, used for programming or debugging the CPLD contents

The pinout file for the EPM240 can be found here at http://bit.ly/1F1ttRk. The two I/O banks spans the following pins: IoBank1 (pin 2..51) and IoBank2 (pin 52..1). Figure 2 shows the EPM240 pins, with IoBank1 colored in light blue, and IoBank2 in light grey. Unfortunately, the different IoBank colors can be hard to distinguish, but that's how it is shown in Altera Quartus2.

I have chosen IoBank1 as the 3.3V

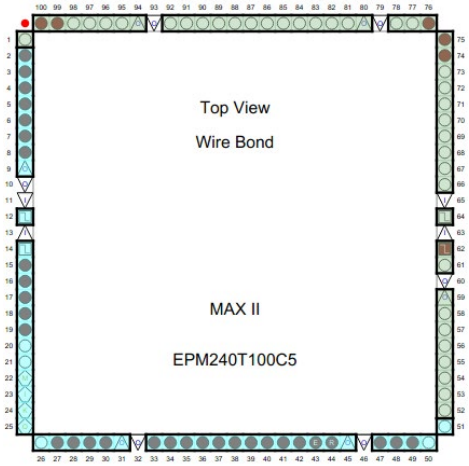


Figure 2 - dedicated (fixed) pins on the EPM240

side (pin2..51) due to the fact that the on-board 3.3V oscillator (OSC) is already connected to GCLK0 (pin12). This way, I didn't have to disable the on-board OSC, and could utilize it later for PWM modulation or a similar purpose.

Due to the need for a 1.8V side and a 3.3V side, there are approximately 40 IO pins on each bank. With 40 pins on the 1.8V side, that means that there are a lot of unused pins. I just needed 4 for SPI and 2 for UART, or 4 for UART if I wanted to connect both UARTs to the 3.3V side, but that is just the way that the EPM240 is made. The larger

Dedicated Pin	100-Pin TQFP
IO/GCLK0	12
IO/GCLK1	14
IO/GCLK2	62
IO/GCLK3	64
IO/DEV_OE	43
IO/DEV_CLRn	44
TDI	23
TMS	22
TCK	24
TDO	25
GNDINT	11, 65
GNDIO	10, 32, 46, 60, 79, 93
VCCINT *(1)*	13, 63
VCCIO1 *(2)*	9, 31, 45
VCCIO2 *(2)*	59, 80, 94
No Connect (N.C.)	-
Total User I/O Pins	80

Figure 3 - IO Bank diagram

EPMs have 4 IoBanks, but there are no cheap boards available. You can read all about the Altera MAX II series in the MAX II Device handbook at http://bit.ly/1GKJfLn.

Modifying the board

As shown in Figure 3, IoBank1 uses pin 9, 31 and 45 to supply the IoBank with the desired voltage, and IoBank2 uses pin 59, 80 and 94. I knew that my CPLD board had 3.3V connected to both IoBanks, so I needed to find a way to separate IoBank2 from the 3.3V that it was currently connected to. This meant that I had to cut some PCB tracks at the relevant positions, and hope that the board was made as a two layer PCB. If it was a four layer board the modification would not be possible, unless I wanted to lift the VCCio pins from the PCB, and solder thin wires to the 0.5mm spaced pins.

I found some pictures of the unpopulated printed circuit board (PCB), and tried to follow the PCB tracks on the pictures, to see where I might be able to cut the tracks for breaking the VCCio2 pins connection to the on-board 3.3V regulator. After spending some time with the pictures and a multimeter, I modified the board as shown in Figure 4 with my X-acto knife.

At first, it didn't seem to succeed, when I just measured the resistance between the C5 pin and the C7 pin. It still appeared to be connected after I have cut the PCB tracks at the two indicated points. I then realized that the VCCio2 pins were connected inside the CPLD, and were measuring very low ohms, even when separated from the 3.3V track. When I measured between C5/C7 and the 3.3V track still connected to C6, I

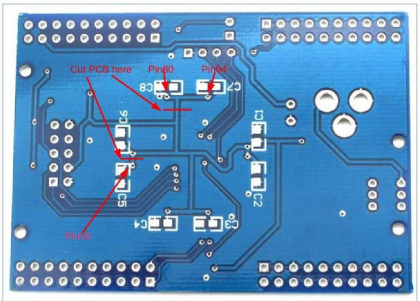


Figure 4 - PCB modifications

could see that the 3.3V connection was disconnected from the VCCio2 pins.

You don't have to cut as deep as I did in the C5 track. I did so because of the above VCCio2 misunderstanding. Also, when I cut the track at C5, I scratched the blue solder mask, and exposed the ground plane. This resulted in an short from C5 plus to the ground plane, when I soldered the wire to C5. Luckily, I measured C5+ to GND before applying power, and discovered my mistake before applying power to the board.

Figure 5 shows the modified PCB, where I soldered a wire from C5 to C7 to connect the VCCio2 pins, and soldered half of a white Dupont wire to C8 . The white Dupont wire (VCCio2 supply) is connected to the 1.8V supply (pin 2) on the U3 IO-Expansion connector.

Next, I soldered a 4-pin single row header onto the unpopulated CPLD CLK header. Note the square to the right, that indicates the GND pin on the CLK header. The GCLK pin header is connected using the following left to right pattern: GCLK1, GCLK2,

GCLK3, GND.

Power

I used 3 wires for the IO-Expansion:

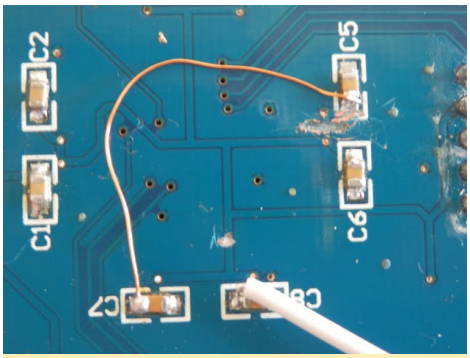


Figure 5 - Modified PCB

1: VDD_IO (pin 2) is connected to the CPLD VCCio2 (white wire on C8)

2: P5V0 (pin 8), is connected to the CPLD board 5V supply

3: GND (pin 7) is connected to GND on the CPLD board

SPI

I used all 4 wires from the SPI:

SPI_1.CLK must be connected to (VHDL – sclk) which is CLK2 on the CLK header

2: SPI_1.CSN must be connected to (VHDL – sel) which is pin 76 on the P1 header

3: SPI_1.MOSI must be connected to (VHDL – mosi) which is pin 75 on the P1 header

4: SPI_1.MISO must be connected to (VHDL – miso) which is pin 74 on the P1 header

I could just as easily have mapped VHDL – miso to pin 73 on the CPLD if I felt that it was a better fit. All of the mapping is done in Quartus2, in either the Pin Planner (easiest), or the Assignment Editor.

UART

I only used two signals from the UART header, since I already had GND via the 5V power connection, leaving the other signals open:

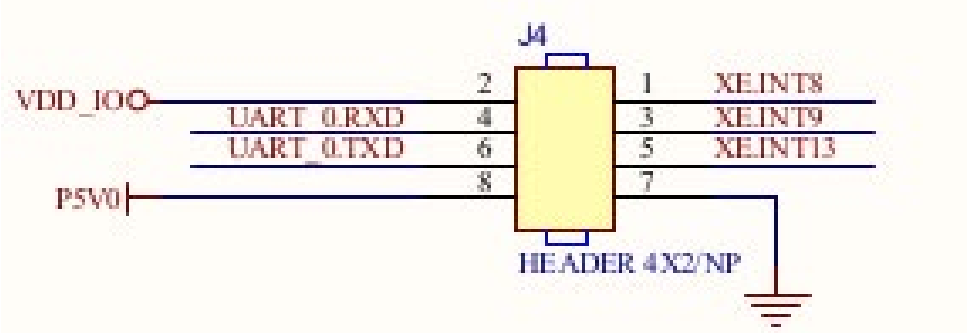


Figure 6 – U3 IO-Expansion connector

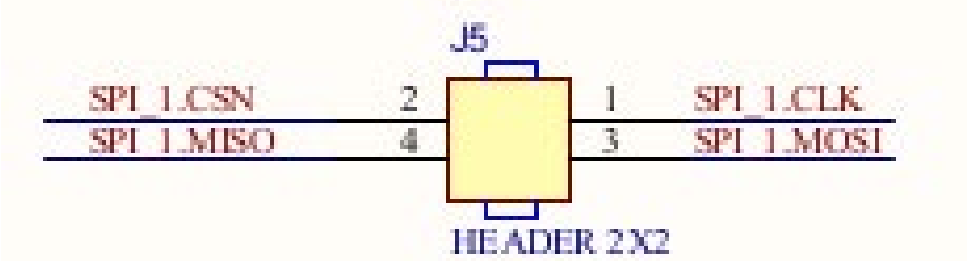


Figure 7 - U3 SPI connector

1: TTA_RXD must be connected to (VHDL - to_u3_rx), which is pin 100 on the P4 header

2: TTA_TXD must be connected to (VHDL - from_u3_tx), which is pin 99 on the P4 header

My 3.3V USB to serial adapter used 4 connections:

1: GND from the ODROID

2: 5V from the ODROID

3: USB TX must be connected to (VHDL - from_usb_tx), which is pin 28 on the P3 header

4: USB RX must be connected to (VHDL - to_usb_rx), which is pin 29 on the P3 header

It only required two extra VHDL lines, and four Pin Planner entries to be able to connect the other U3 UART on the Expansion header to the CPLD and another USB-Serial adapter, which is the benefit of using a CPLD.

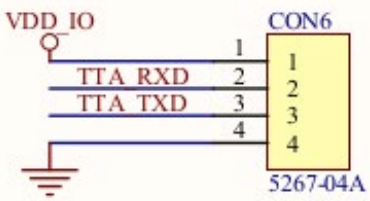
Installing Quartus2

I am not going to cover installation and use of Quartus2, since there are a lot of places on the Internet where you can find installation guides for it. One of them is the Pyro course that I mentioned in the beginning of the article.

I am using the 64-bit Linux edition of the free Quartus2 Web Edition on my Linux Mint 17 machine, but recommend that most people use the Windows version, since the Linux version usually needs some additional libraries which must be manually built, along with some Quartus2 files that need to be modified.

The most recent version of Quartus2 that has MAX II support is Quartus2 v.13.0sp1. Do not install a newer version, as it will not work properly. To download it, visit http://bit.ly/1cdWtJg, which requires registration. Select the Combined files tab, and download the

Figure 8 - U3 UART header



4.4GB file. Install Quartus, register at Altera, and get a free web license file.

Building the design

Download the quartus project file from http://bit.ly/1H8iGQs, then extract the downloaded project, and start Quartus2. Select File->Open Project, browse to the extracted project, and select the gz_16o8i.qpf file. Build the design by selecting the button marked with the red ring, as shown in Figure 9, and ignore the warnings.

Next, shut down the ODROID, and disconnect the power. Connect the Altera USB Blaster 10-pin cable to the CPLD Board and connect 5V and 1.8V to the CPLD board. I used +5V and GND from the ODROID board, and connected them to the P1 header on the CPLD board. Then I connected the 1.8V from the ODROID to the modification point on the CPLD board.

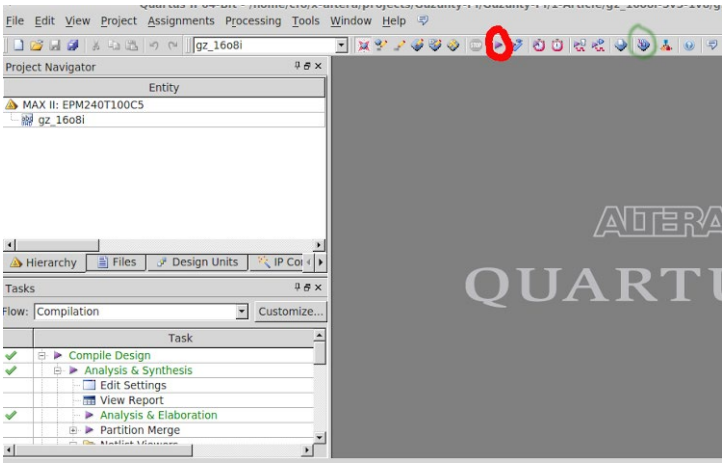
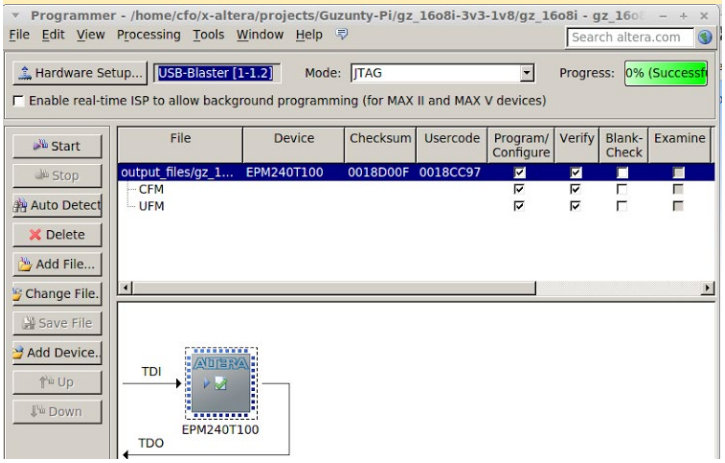


Figure 9 - Quartus 2

Power on the ODROID, connect the programmer to the PC's USB port, and start the programming tool by pressing the button indicated by the green ring in Figure 9. Press the Hardware setup button, and select the jtag programmer. You should see something similar to that shown in Figure 10. Press Start, and program your design file into the CPLD. If it shows the Green 100% successful, as seen in Figure 10, the CPLD is now ready for use.

Figure 10 - Quartus Programmer



Preparing the software

The Guzunty Pi uses SPI for the communications, and in order to use fast SPI, we need to load the ODROID-U3 SPI kernel module. I recommend that you use the tool from the two ODROID web pages at http://bit.ly/1JmO6HH and http://bit.ly/1RS94lw, as they describe how to load the kernel module, as well as how to update the U3 software if needed.

Load SPI module

One can load the SPI kernel module manually by typing the following into a Terminal window:

```
$ sudo modprobe spi-s3c64xx
```

However, it would be better for it to be automatically available on every boot. While logged in as root, edit the file /etc/modules and add the below lines to the end of the file:

```
# SPI for the Guzunty Pi CPLD
spi-s3c64xx
```

It's considered good programming practice not to require root privileges when accessing SPI devices. To grant access to other users, create a new spidev system group, then set up the udev system to give the spidev group read and write (r/w) access to the SPI devices:

```
$ sudo groupadd -f --system spidev
$ sudo usermod -a -G spidev username # replace user-
name with your ODROID username
```

Next, make a udev rule for spidev, allowing access to members of the group spidev. As root, create a new file called /etc/udev/rules.d/99-spidev.rules, which is shown below:

```
# /etc/udev/rules.d/99-spidev.rules
# Allow access for the group members of spidev, in
order to be able to access SPI as a normal user
SUBSYSTEM==”spidev”, MODE=”660”, GROUP=”spidev”
```

After saving the file and rebooting, members of the spidev group will have proper access to the spidev subsystem. The SPI modules may be checked with the following command:

```
$ lsmod | grep spi
spideV 5641 0
spi_s3c64xx 9849 0
```

Then, verify that the group spidev has proper access to the SPI device:

```
$ ls /dev/spi* -l
crw-rw---T 1 root spideV 153, 0
Jan 1 2000 /dev/spidev1.0
```

Guzunty software

The Guzunty software consists of two sections, each in its own directory:

1: gzlib, which is the Linux library that interfaces with the SPI subsystem

2: The user program that matches the CPLD design, which is gz_16o8i in this case

In order to get the Guzunty software running, it's necessary to install the Linux packages git and build-essential. Git is used for cloning the Guzunty software repository, and build-essential is the Linux build system that provides the C compiler and other build tools:

```
$ sudo apt-get install git \
  build-essential
```

Create a Guzunty directory, then clone the Guzunty repository into the Pi subdirectory:

```
$ mkdir Guzunty
$ cd Guzunty
$ git clone https://github.com/\
  Guzunty/Pi.git
```

It's necessary to change the following line in the Guzunty library source file called gz_spi.c:

```
$ cd Pi/src/gzlib/src/
```

Edit the gz_spi.c file, and change the device named "/dev/spidev0.0" to "/dev/spidev1.0" and save the file:

```
void gz_spi_initialize() {
// spi_open("/dev/spidev0.0");
// Comment out the
Raspberry Pi line by adding // in
front of it
spi_open("/dev/spidev1.0");
// Add the correct odroid SPI
device above
initialized = 1;
}
```

I changed the SPI speed from 10MHz to 1MHz, since 10MHz was too fast for my Saleae Basic logic analyzer:

```
//#define SPI_MAX_SPEED 10000000
// 10 MHz
#define SPI_MAX_SPEED 1000000 //
1 MHz
```

You don't need to change the speed to 1MHz of you don't want to check the signals with a Saleae, or have a faster logic analyzer. I think that the ODROID-U3 is capable of running 40MHz on the SPI system, but I have only tested it at 10MHz. If you choose 40Mhz, you will have to keep all the SPI wires as short as possible, or you will run into problems with signal integrity. If you have problems, try lowering the SPI speed to 1MHz, and see if that solves it. If it does, you probably used wires that were too long.

Next, build and install the Guzunty SPI library, which will then be ready to be linked along with our user/CPLD design program:

```
$ make
$ sudo make install
```

Since our design is based on the Guzunty gz_16o8i design with 16 outputs and 8 inputs, change directory to Pi/src/gz_16o8i and build the user program:

```
$ cd Pi/src/gz_16o8i
$ make
```

If you get an errors that says "fatal error: gz_spi.h: No such file or directory", you forgot to perform the "sudo make install" of the gzlib. Otherwise, your program and CPLD are ready to use. Start it with the following command.

```
$ ./gz_16o8i
```

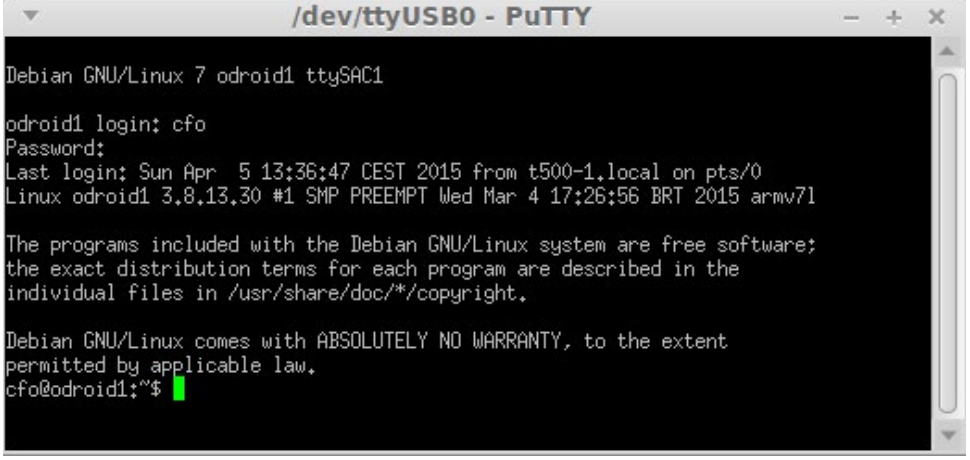
Then, test out the UART connection from the host computer using PuTTY, which, if successful, should present a console as shown in Figure 11.

Conclusion

Through the use of Guzunty Pi, I achieved my goal of creating a cheap, versatile level converter for use with by 3.3V USB to serial converter. This method ended up being more complicated than just using the TI chip, but I also think that Guzunty for the ODROID-U3 is a nice addition to a powerful board.

For questions and comments, please refer to the Guzunty thread on the ODROID forums at http://bit.ly/1d5qqM4. I would like to thank forum members @odroid and @robroy for helping me out during development.

Figure II - Guzunty Pi console via UART



ANDROID DEVELOPMENT

CREATING A CUSTOM WEB SERVER SERVICE

by Nanik Tolaram

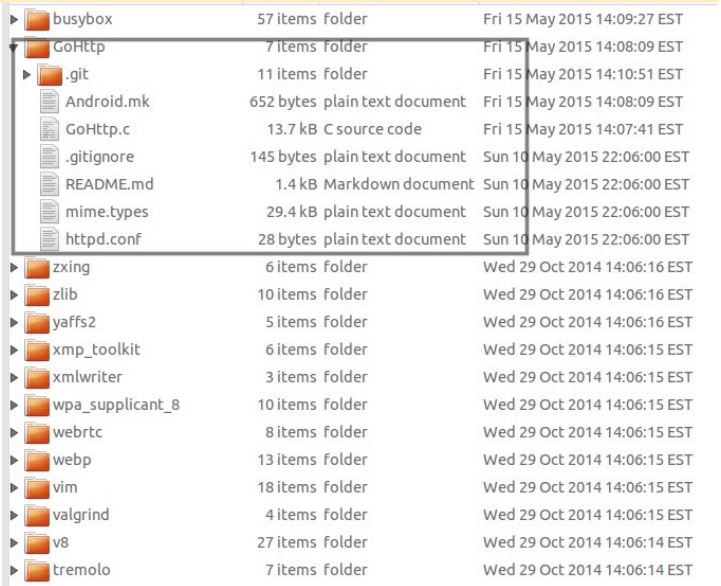
Previously, I examined how Android boots up, along with the various services that are launched during the boot process. I also explored the contents of the init.rc which showed the different services that are required to make Android work.

In this article, I take a look at how to add a custom service using a web server as an example. This is accomplished in two steps: adding a native app as part of the init process, then porting a Linux native app to Android.

GoHttp

We will use an open source web server as part of this exercise. The web server is a very basic file serving application and is not a full fledged web server. The ported application can be downloaded from from https://github.com/nanikjava/GoHttp from the master-android-patch branch. The original project is also available for reference at https://github.com/fekberg/GoHttp.

Figure I: Gohttp inside external/ folder



Build file

Since we want to build the application as part of our Android image, we will need to embed it into the Android source code. As shown in Figure 1, you need to place it into the external/ folder.

The first step of porting is to create the Android.mk build file, which is similar to the Linux Makefile. The Android.mk file for the GoHttp project used in this example should look like this:

```
LOCAL_PATH:= $(call my-dir)

include $(CLEAR_VARS)
LOCAL_SRC_FILES :=  GoHttp.c
LOCAL_LDLIBS += -lrt -ldl -lpthread -llog
LOCAL_CFLAGS := -DDEBUG_ANDROID
LOCAL_SHARED_LIBRARIES := liblog
LOCAL_MODULE := gohttp
include $(BUILD_EXECUTABLE)

include $(CLEAR_VARS)
LOCAL_MODULE := httpd.conf
LOCAL_MODULE_CLASS := ETC
LOCAL_MODULE_PATH := $(TARGET_OUT)/etc
LOCAL_MODULE_TAGS := optional
LOCAL_SRC_FILES := $(LOCAL_MODULE)
include $(BUILD_PREBUILT)

include $(CLEAR_VARS)
LOCAL_MODULE := mime.types
LOCAL_MODULE_CLASS := ETC
LOCAL_MODULE_PATH := $(TARGET_OUT)/etc
LOCAL_MODULE_TAGS := optional
LOCAL_SRC_FILES := $(LOCAL_MODULE)
include $(BUILD_PREBUILT)
```

The Android.mk is similar to other projects inside the external/ directory. Notice that there are 2 files being copied as part of the build process:

- httpd.conf → contains the configuration for the webserver
- mime.types → contains the file types that are allowable by the webserver app

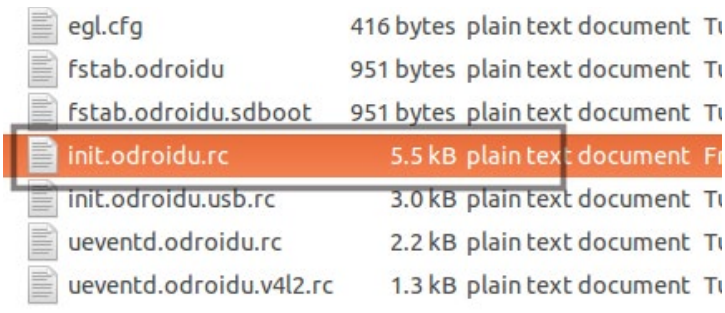
Figure 2: gohttp executable

The webserver is configured to listen at port 8888, which is declared inside the httpd.conf file.

The other item that need to be modified inside the app is the logging, since it make sense to make sure that the app sends any logging information to the logcat service. Here is the new code that is added inside GoHttp.c that will do the logging:

```
#ifdef DEBUG_ANDROID
#include <android/log.h>
#define LOG_TAG "gohttp"
#define PRINT(...) __android_log_print(ANDROID_LOG_
INFO, LOG_TAG, __VA_ARGS__)
#else
#define PRINT(...) fprintf(stdout, "%s\n", __VA_
ARGS__)
#endif
```

Figure 3: init file for ODROID-U3



Once you have successfully compiled the ODROID-U3 Android source code, you will see a gohttp executable file inside the out/target/product/odroidu/system/bin directory, as shown in Figure 2.

Initialization

The last step of the porting is to run the gohttp application as part of Android startup process. In order to do that, we will need to modify the file called init.odroidu.rc inside device/hardkernel/odroidu/conf directory, as shown in Figure 3.

```
on post-fs-data
    mkdir /data/media 0770 media_rw media_rw

    setprop vold.post_fs_data_done 1

    mkdir /data/www  0770 system system
```

Figure 4: mkdir for creating directory

We also need to add a service to start gohttp:

```
service gohttp /system/bin/gohttp
    class core
```

Under the "on post-fs-data" section, you need to add the statement indicated in Figure 4. This statement is used to create the directory where you will put the .html files that you want to make available for the user to access.

Once you boot up your ODROID-U3, you can then copy an index.html file to /data/www folder and access it via the server by using any browser to navigate to the local site at http://<YOUR_ODROID_U3_IP>:8888/index.html.

If you'd like to learn more about Android programming, contact Nanik directly, or follow his latest posts, please visit his website at http://naniktolaram.com.

With Nanik's help, you're now fully equipped to create your own custom Android service

ULTRASTAR DELUXE KARAOKE

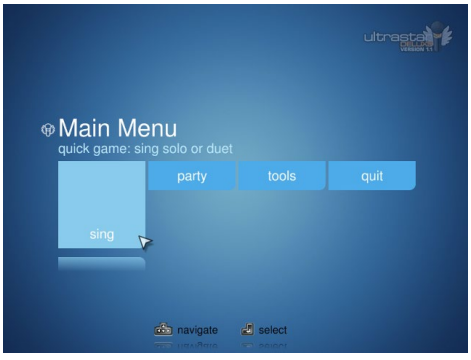
BECOME AN ODROID ROCK STAR

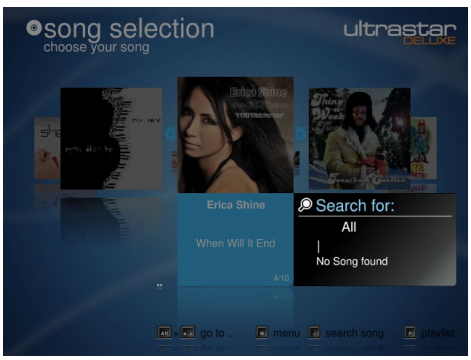
by @vOltumna

Are you an aspiring rock star? UltraStar Deluxe is a free open-source karaoke game, similar to the Sony PlayStation game called SingStar, that allows you to create a portable karaoke machine. The source code is published at http://bit.ly/1e9amZW, which can be easily compiled for the ODROID, turning it into the ultimate party machine! I use two wireless SingStar microphones which work great.

Compile ffmpeg

Ultrastardx has customizations for different versions of ffmpeg in order to play media. However, I found during my testing that not all existing releases of ffmpeg worked properly. I tried different versions and got different errors when I compiled ultrastardx, but had the most success with ffmpeg version 2.1.5. Before you install it, it is a good idea to remove your old version and also some dev packages, otherwise Ultrastar might find false version information during compilation.






```
$ sudo apt-get remove ffmpeg
libavutil-dev \
  libswscale-dev libavcodec-dev
libavdevice-dev
```

Then, install the custom version:

```
$ wget http://www.ffmpeg.org/re-
leases/ffmpeg-2.1.5.tar.bz2
$ tar xf ffmpeg-2.1.5.tar.bz2
$ cd ffmpeg-2.1.5
$ export CFLAGS="-O3 -D__ARM_
NEON__ -fPIC \
  -march=armv7-a -mfloat-abi=hard
-mfpu=neon \
  -ftree-vectorize -mvectorize-
with-neon-quad \
  -mcpu=cortex-a9 -mtune=cortex-
a9"
$ export CXXFLAGS="-O3 -D__ARM_
NEON__ -fPIC \
  -march=armv7-a -mfloat-abi=hard
\
  -mfpu=neon -ftree-vectorize
-mvectorize-with-neon-quad \
  -mcpu=cortex-a9 -mtune=cortex-
a9"
$ ./configure --enable-libvorbis
--enable-pthreads \
  --enable-libmp3lame --enable-
nonfree \
  --enable-gpl --enable-libxvid
--enable-libx264 \
  --enable-shared --prefix=/usr
$ make -j5
$ make install
```

On Debian or Ubuntu, you may want to run checkinstall -D in order to create an installable .deb file.

Compile ultrastardx

Ultrastardx is written in Object Pascal, so you will need the Free Pascal Compiler and some dependency units:

```
$ apt-get install fp-compiler fp-units-misc fp-units-base \
  fp-units-math fp-units-fv fp-units-fcl
```

Then, download the latest source of ultrastardx and configure it:

```
$ svn checkout \
  svn://svn.code.sf.net/p/ultrastardx/svn/trunk \
  ultrastardx-svn
$ cd ultrastar-svn
$ ./configure
```

Because of a compiler error, the compilation won't work until the optimization "-O2" is removed on the PFLAGS_RELEASE_DEFAULT option in src/Makefile in line 102. To get it working with the OpenGL wrapper, you also need to remove or comment lines 4323 to 4330 in the file src/lib/JEDI-SDL/OpenGL/Pas/glext.pas:

```
//   @glCopyTexSubImage3D := SDL_GL_GetProcAddress('glCopyTexSubImage3D');
//   if not Assigned(glCopyTexSubImage3D) then Exit;
//   @glDrawRangeElements := SDL_GL_GetProcAddress('glDrawRangeElements');
//   if not Assigned(glDrawRangeElements) then Exit;
//   @glTexImage3D := SDL_GL_GetProcAddress('glTexImage3D');
//   if not Assigned(glTexImage3D) then Exit;
//   @glTexSubImage3D := SDL_GL_GetProcAddress('glTexSubImage3D');
//   if not Assigned(glTexSubImage3D) then Exit;
```

Then, compile the program:

```
$ make
$ make install
```

As before, you may want to run checkinstall -D in order to create an installable .deb file if using Debian or Ubuntu.

If you receive errors about the versions of libavutil, export those versions using environment variables and configure again. Make sure to remove the optimization in PFLAGS as noted above.

```
$ export libavutil_VERSION=52.48.101
$ export libavcodec_VERSION=55.39.101
$ export libavformat_VERSION=55.19.104
$ export libavdevice_VERSION=55.5.100
$ export libavfilter_VERSION=3.90.100
$ export libswscale_VERSION=2.5.101
$ export libswresample_VERSION=0.17.104
$ ./configure
```

You may now start ultrastardx, but it will be a bit slow, because OpenGL has to be emulated by software. This may be fixed by adding an EGL wrapper.

EGL configuration

Starting ultrastardx with the wrapper often gives me an error about creating the EGL display, but it sometimes works without any change in configuration. Type the following commands to download and compile the wrapper:

```
$ git clone git://github.com/lunixbochs/glshim
$ cd glshim
$ cmake .
$ make GL
$ git clone git://github.com/lunixbochs/glues
$ cd glues
$ cmake .
$ make
```

Copy the dynamic libraries to /usr/local/lib. Usually, you just have to export the LD_LIBRARY_PATH to this folder and start the program. However, this does not always work, so here is my start script which sometimes requires between 5 and 10 attempts. When it works properly, it provides a very stable and fast game experience:

```
#!/bin/bash
export LD_LIBRARY_PATH=/usr/local/lib
while true
  do
  ultrastardx
  if [ "$?" -ne 0 ]; then
    break
  fi
done
```

Game configuration

Here is the graphics part of my configuration file config.ini that seems to work the best:

```
[Graphics]
Screens=1
FullScreen=On
Visualization=Off
Resolution=640x480
Depth=16 bit
TextureSize=256
SingWindow=Big
Oscilloscope=Off
Spectrum=Off
Spectrograph=Off
MovieSize=Full [BG+Vid]
VideoPreview=On
VideoEnabled=On
```

Compilation notes

If the compilation does not complete successfully, you may have to install some development packages:

```
$ apt-get install \
  libsqlite3-dev \
  portaudio19-dev \
  libsdl-image1.2-dev
```

After compiling and installing, start Ultrastar with the following script:

```
#!/bin/bash
export LD_LIBRARY_PATH=/usr/lo-
cal/lib
ultrastardx
```

If you have questions, comments, or suggestions, please refer to the original post at http://bit.ly/1bYvbp7.

TEKKEN 6
THE ULTIMATE FIGHTING GAME

by Justin Lee

Tekken 6 is a very popular martial arts fighting game that runs great on any ODROID. You can check out Hardkernel's Tekken 6 PPSSPP gameplay video at http://bit.ly/1f1BqdX. To play Tekken 6 on your ODROID, first download and install the latest Android image for your device, then follow these steps:

1. Change HDMI resolution to 1280x720 HD using the ODROID-Utility
2. Change the CPU Governor to Performance using the ODROID-Utility
3. Install the PPSSPP application, one of the best PSP emulators available
4. Configure the PPSSPP Settings:

Graphics menu
Rendering Mode: Non-buffered rendering (Speedhack)
Simulate block transfer (unfinished): Check
Framerate control
Frameskipping: 2
Auto frameskip: Check
Prevent FPS from exceeding 60: Check
Alternative speed: 0

Features
Postprocessing shader: Off
Strech to display: uncheck
Small display: uncheck
Immersive mode: Check

Performance
Rendering resolution: 1xPSP
Display resolution (HW scaler): 2xPSP
Mipmapping: Check
Hardware transform: Check
Software skinning: Check
Vertex cache: Check
Lazy texture caching: Check
Retain changed textrue: Check
Disable slower effects: Check

Hack settings
Disable alpha test: Check
Texture coord speedhack: Check
Show FPC count: Both

Controls menu
On-screen touch controls: Uncheck

System menu
Multithreaded: Check

Tekken 6 combines fast action with awesome martial arts moves

MEET AN ODROIDIAN

MARKHAM THOMAS (@MLINUXGUY) A HIGHLY EXPERIENCED AND WELL-TRAVELED LINUX EXPERT

edited by Rob Roy

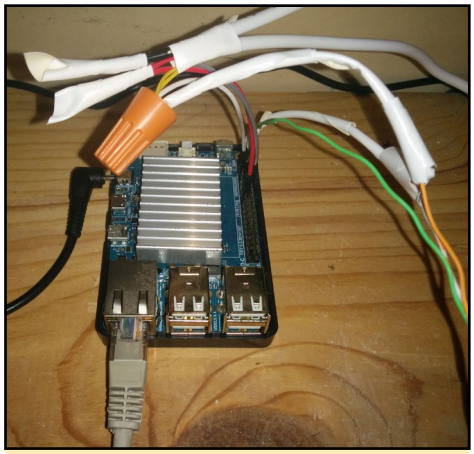
From coding to poking black box issues with sticks, you can count on Markham to tackle all sorts of problems!

Please tell us a little about yourself.

I live in central Oklahoma, and work in the technology field with people from all over the world. We communicate in chat rooms, virtual rooms, and voice conferences. I have 3 cats who think that keyboards are meant to be walked on, and that warm ODROIDs are to be slept on, as well as a significant other. We live in a custom-designed home with unique features such as a commercial metal roof, geothermal heating and cooling, and a design from my architect father resembling something that Frank Lloyd Wright would have done. It has lots of hidden spaces, ramps, and firdowns for cats to climb.

How did you get started with computers?

I bought my first computer at age 14 with money saved from hauling hay in the fields. It was a TRS-80 Model III. My mother told my grandmother that her son had blown $1400 that he should have saved for college on a calculator. Within a few years, I had tricked out that TRS-80 with more RAM (64k) and dual-disk drives, and hacked it to overclock to 5Mhz with a switch on the side. I still recall the first prompt I got upon powering it up: "Cass?" I had no idea what that meant (cassette tape loader), but by the time I was in college, I was writing Z80 assembler code for it. Magazines at the time provided example code and circuits, which is where you got the specs on how to hack the systems, since there was no Internet.



An ODROID-C1 monitoring geothermal temperatures in Markham's basement

That Z80 assembler skill got me a job during University where we built, and I coded, giant hydraulic test stands for aerospace and heavy industry. The hex codes were entered by hand into EEPROMs that were 2KB in size. I can still recall the hex codes for many Z80 instructions after typing so many of them into the programmer. Later, we created hardware to link the CP/M build system to the EEPROM programmer and did away with hand entry.

After college, I went to work for a major technology company and continued working with the latest computer systems, but it was rare to re-experience the thrill of exploring those early systems, where you could trace circuits and make them do things through hacking that were never intended.

What drew you to the ODROID platform?

For years, I built a new home PC every 18 months, but by the time Windows 7 came about, I had lost interest in upgrading and tricking out PCs, and started looking for smaller systems that I could run Linux on, especially ones that didn't over-heat my office. I had been investigating ARM processors with the idea of building a Linux-capable board when the Raspberry PI came out. I immediately got four of them and started pushing them to their limits. I hit those limits pretty quickly, so while doing projects with the Raspberry Pi, I kept looking for a better board. Once I discovered the ODROID-X, I immediately got one and started a deep dive into its abilities.

What make the ODROID computers so much more interesting to me than competing boards is the dynamic community and support. The Raspberry Pi community is large, but they sometimes lack the fast dynamic responses you get with the smaller ODROID community. It was fascinating to see the capabilities of the Odroid-X increase after release, and captured again some of that thrill

of participating in a revolution. This is a revolution, just like the PC revolution decades ago.

Which ODROID is your favorite?

I would have to say the ODROID-C1 is my favorite, I have five of them, and only one of each of the other ODROIDs. I like it because of the vast amount of code that can be easily adopted from the Raspberry Pi world. Its CPU is based on a smaller geometry than many of its competitors, giving me a fanless low-power fast Linux box that is cheap enough that I can send it up on a kite to a thousand feet, or stick it up on the roof as a weather station, and not care if it fails in a few years.

Describe your ODROID setup and how you use it.

It will be hard to pick just one C1 project to discuss. My home office is lit with blinking lights from all of the ODROIDs and other boards. I have a Linux development board where I hack the kernel and its drivers, and another one that is dedicated to testing various expansion boards.

I also have one in the basement that measures the Geothermal loop temperatures, and one on the roof with an Arduino piggy-backed on it, which reads weather station data. Probably the most interesting ODROID is the device that is measuring the Geothermal data. I monitor both the loop inlet and outlet temperatures, outside temperature, solar brightness and ultraviolet (UV) levels, and build graphs from that data to help me interpret the efficiency of my ground loop as environmental conditions vary.

Most of my ODROIDs are headless and run Ubuntu. I have the development one on a monitor so that I can reach it after I destroy the networking or break the kernel. One Raspberry Pi is dedicated to running a console UART serial port link under the screen command so that I can SSH into the Pi and see what I did that killed the ODROID-C1 kernel.

I have another C1 that replaced a Pi that was as a FM radio recorder. That box controls a low-cost FM tuner board via SPI, and feeds the audio into a USB sound card in order to record various public radio programs to MP3 files. I then load them into a player to listen to while I run. I could just download the MP3 from them, but it's saved automatically to my network share. The C1 has more than enough performance to record it at higher bit-rates.

You are very generous on the ODROID forums with sharing your knowledge of hardware, electrical engineering and Linux programming. How did you become so proficient?

I've been doing this for a long time, but that alone doesn't make me proficient or necessarily good at it. I think what it takes is a love of tearing apart things to see what makes them tick. Although I majored in Electronics in college, I never really used it. Instead, I mostly spent my time on the software side, debugging black box problems.

For this computer revolution, I want to dive deeper into the technology, so I usually pick some feature that I'm interested in, such as network performance, and start tearing apart how it works and how I could improve it. I then start testing changes.

A long time ago, I wrote SVGA drivers for DOS in assembly language, where I had to count each instruction cycle and optimize it. That experience has been invaluable for me when I'm improving code performance. There's nothing quite like the feedback of instantly seeing your optimizations improve a line-draw routine on the screen.

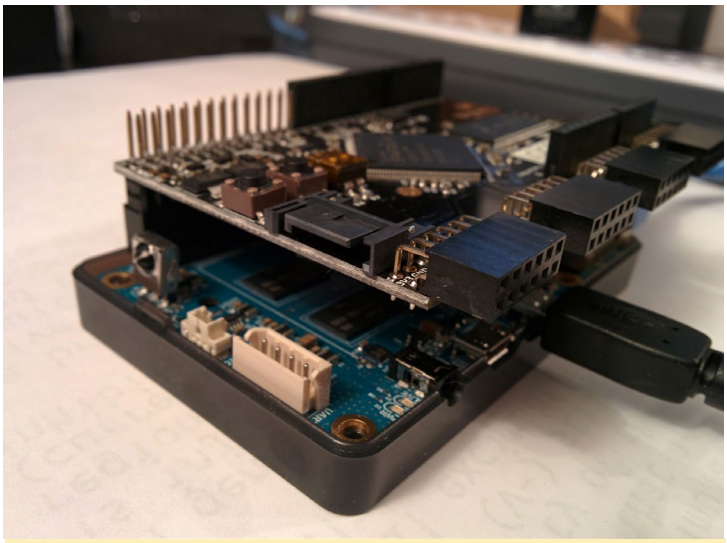
My engineering background is now playing a bigger part because, after spending years analyzing black box problems on Linux and Unix (usually high level code or performance issues), I've started working with FPGAs, and now finally have the complete picture of how all the hardware components interact to make a system.

What hobbies and interests do you have apart from computers?

I spent years as a runner, but have cut back in order to play tennis, a sport that means playing in extreme conditions in Oklahoma. You rarely see professionals on TV chasing their own ball across the court like we do here.

I have hiked and climbed most of the bigger mountains in the southwest United States. I like to snow ski, and have a 18 foot catamaran that I need to refurbish before taking out again. When I hike, I always have camera gear with me that weighs more than my backpack, but now I'm mostly doing aerial photography from either a kite or one of my drones.

One project that I still have on my to-do list is sending an ODROID up with the kite, and coding some image stabilization routines for a 9 axis mod-



LogiPi FPGA board connected to an ODROID-C1, used for learning FPGA programming

Markham enjoying a long hike in beautiful Yellowstone Park

ule. Currently, I just have it shoot pictures every 30 seconds and pick out the best ones.

I am a member of the Planetary Society, and have helped to fund many of their projects, with their Light-Sail being one of the latest. I tend to follow developments in areas like CubeSat and other projects that bring space access into the realm of the maker community.

Are you involved with any other computer projects unrelated to the ODROID?

I have been working with multiple FPGA boards including ones with Linux on-board, such as my Parallella board. However, I don't have any particular project planned with them yet beyond just learning digital circuits.

I have all the parts in my basement workshop to make nine Infinite Noise TRNG boards. I'll do some tests once I get one built in order to compare it to the LSFR random number generator on the ODROID-C1.

I have a MakerBot 2 that I have hacked with an aluminum heated build plate and aluminum arms, side covers and top. I use it to print cases for my various ARM boards, as well as support structures for various projects such as a 3-axis stabilized camera platform for the kite.

I also have been making home-made PCB boards using my LaserJet and a laminator. I have a concentrator board for cleaning up the wiring in my rooftop weather station that I'm laying out, and will then etch to upgrade that.

The next big project will be taking my 25HP diesel tractor and replacing its gauges with trans-reflective LCD panels driven by an Arduino. My nephew wants to turbo-charge it so it will need additional sensors.

What type of hardware innovations would you like to see for future Hardkernel boards?

The next big jump in ARM development boards that I'm anxiously awaiting is the arrival of the 64-bit boards. I would love for Hardkernel to release one, but until that happens, there are features missing now that would be useful.

The inclusion of a 1GB network interface in the C1 was a nice boost, however, the addition of a high-speed port such as SATA or USB 3.1 would allow more projects. One innovative feature found on the Beaglebone is their Programming Real-Time Unit (PRU), and something similar for a Hardkernel board might be an Arduino chip or the traces to add one yourself.

The expansion market and open platform architecture was one of the things that drove the PC revolution, so having the ODROID-C1 around with its Raspberry Pi-compatible expansion header immediately improves the value of the Hardkernel board. I would like to see future boards have a similar header, even if it is only the pre-B+ header.

A high speed expansion interface would also be nice, but from a cost perspective, perhaps the new USB 3.1 specification would give enough throughput and support for fast peripherals. That would be an ideal addition to new boards in the Hardkernel lineup.

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

There are many resources out on the Internet freely available for anyone who wants to learn more about programming. Don't let your unfamiliarity with coding or a particular language hold you back. Pick a project, choose your language, and start searching the Internet for a bit of skeleton code to get you started. Once you have the simple outline code, start fleshing it out by constantly referring to examples. I have a 4K monitor, and often have 10 or more browser windows open for reference when coding.

I recommend picking a favorite open-source project and adding a feature to it, which you may be able to get accepted upstream in the project. GitHub is a great resource for finding such projects, and you can either fork it or just contribute features.

The tricky thing about just knowing how to program without understanding how microprocessors work is that you can end up writing inefficient code. Take some time out to profile your application, and include mixed source and assembler when you profile it, so that you can see how your code gets implemented by the compiler.

If you really want a deep understanding of how computer hardware works, like bit tests, bit shifts, and addition, the best approach would be to learn boolean logic and take a few online FPGA courses. Then, dig into some of the code on http://opencores.org to see how computer circuits really work.