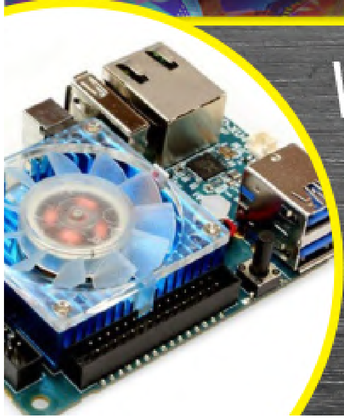
oDrive Car • Amiga Emulation Box • Xbox on ODROID-H2 • Gamestation

ODROID Magazine

Year Seven
Issue #76
Apr 2020

THE HARDKERNEL AND ODROID MAGAZINE TEAMS PRESENT A SPECIAL GO ADVANCE ISSUE TO HAVE FUN WHILE YOU:

STAY SAFE AT HOME



WEBTHINGS: ARMBIAN ODROID-XU4 FOR INTERNET OF THINGS

ODROID-GO • RETRO-GO: A SPIFFY ODROID-GO EMULATOR LAUNCHER

SDL2 SCREEN ROTATION: HYPERCHARGE YOUR ODROID-GO ADVANCE'S GRAPHICS

SDL2 With Screen Rotation: Hypercharge Your ODROID-GO Advance's Graphics

April 1, 2020

I have successfully ported SDL2 with internal screen rotation, so you don't need a modified build of PPSSPP or Emulationstation anymore.

Amiga Emulation Box: Turn Your ODROID-XU4 Into a 1980s Wonder Computer

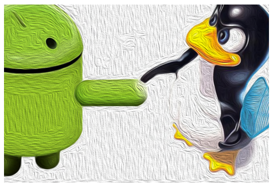
April 3, 2020

This tutorial will show you how to set up an Amiga emulator on your ODROID-XU4 by building your own copy of Uae4Arm and setting it up with kickstart ROMs and Workbench files from Amiga Forever or wherever you got your copy from. This tutorial assumes you have a working ODROID-XU4 ▶️

BMW oDrive Car PC

April 1, 2020

I have been working on this project for almost a year now and feel confident enough now to present it to the public - the almighty iDrive supplemental solution for all people who do not have interest in fitting a 1500€ retrofit solution.



Running Linux Under Android

April 1, 2020

The ODROID-N2 seems like a perfect Android TV system - everything runs smoothly and converts a regular TV into a valuable smart TV. All is well if that is the only thing you want to do with your ODROID-N2, but for me, it is not enough. I wanted to use ▶️

Shall We Play A Game? - Nintendo Drags Pokémon Into the 21st Century...Kicking and Screaming

April 1, 2020

Imagine being able to share, trade, and move your Pokémon between your games on different devices via a centrally-located, Internet-connected server. Sounds like some really wacky, far-out, futuristic stuff, eh? Now you can with pokémon home.

Building An Xbox Using an ODROID-H2

April 1, 2020

The OG Xbox rocked our world - we still call the media player 'the xbox' to this day. After it was modded it became a media powerhouse that was way ahead of its time. And now it is time to emulate it itself!

Retro-Go: Yet Another ODROID-GO Emulator Launcher

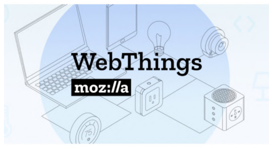
April 1, 2020

Retro-Go is a launcher with NES/GB/GBC/SMS/GameGear/Colecovision emulators. The emulator's code is based on Go-Play "Triforce" and the launcher's design is strongly inspired by pelle7's; however, it isn't a fork.

Retro Roller: An Optimized Gaming Experience For Your ODROID-GO Advance

April 1, 2020

Retroroller is a pre-built image for the ODROID-GO Advance that provides RetroArch 32bit and 64bit on CrashOverride's one and only stock image. Among other tweaks, a custom kernel is integrated that supports sleep. The idea with this image is to be able to update via a rolling package release instead ▶️



WebThings on Armbian: Using the ODROID-XU4 for the Internet of Things

April 1, 2020

Mozilla WebThings is a platform for controlling home devices.

Modding Your ODROID-GO Advance for Wireless Charging: A Simple DIY Project

April 1, 2020

I added wireless charging capabilities to my ODROID-GO Advance.

Linux Gaming: GameStation Turbo Advance

April 1, 2020

Here's what I call ODROID GameStation Turbo Advance. A feature packed gaming OS for the ODROID-GO Advance utilizing X11 as a graphics backend and attract mode as a frontend for emulators, games and tools.

SDL2 With Screen Rotation: Hypercharge Your ODROID-GO Advance's Graphics

April 1, 2020 By @AreaScout Gaming, ODROID-GO Advance

I have successfully ported SDL2 with internal screen rotation, so you don't need a modified build of PPSSPP or Emulationstation anymore. SDL2 will report a screen resolution of 480x320. There are tons of free games to find on GitHub, and probably some good emulators using SDL2. There are also some good open source GUI's, such as nanoguisdl, which support SDL2 too.

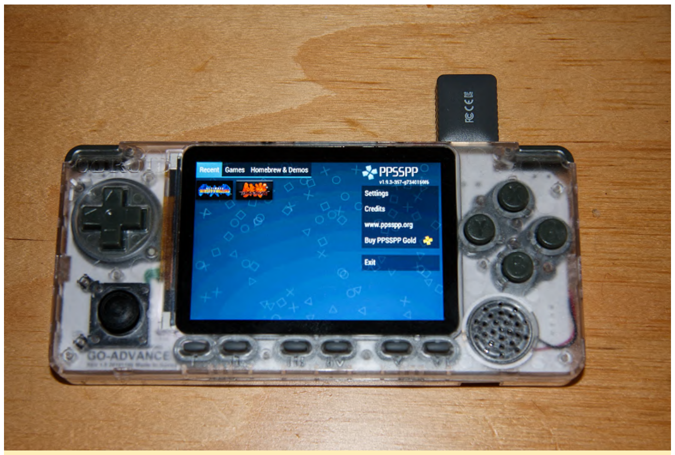


Figure 1 - ODROID-Go Advance running PPSSPP using SDL2 with screen rotation

Preparation

First, you will need the Mali GPU driver as as a deb package, so that it won't be overwritten by the Mesa libraries:

```
$ wget
https://oph.mdrjr.net/meveric/pool/go2/libm/libmal
```

```
i-rk/libmali-rk-bifrost-g31-rxp0-gbm_1.7-
1+deb10_arm64.deb
$ wget
https://oph.mdrjr.net/meveric/pool/go2/libm/libmal
i-rk/libmali-rk-dev_1.7-1+deb10_arm64.deb
$ sudo apt install ./libmali-rk-dev_1.7-
1+deb10_arm64.deb ./libmali-rk-bifrost-g31-rxp0-
gbm_1.7-1+deb10_arm64.deb
```

Installation

The first package is the runtime libraries, and the second is the development package which contains headers, in case you are about to compile applications. The SDL2 library will install in /usr/lib/aarch64-linux-gnu and will overwrite any previously installed SDL2 library.

```
$ wget https://www.areascout.at/libsdl2-2.0-
0_2.0.10+dfsg1-1ubuntu1_arm64.deb
$ wget https://www.areascout.at/libsdl2-
dev_2.0.10+dfsg1-1ubuntu1_arm64.deb

$ sudo apt install ./libsdl2-2.0-0_2.0.10+dfsg1-
1ubuntu1_arm64.deb
$ sudo apt install ./libsdl2-dev_2.0.10+dfsg1-
1ubuntu1_arm64.deb
```

For comments, questions, and suggestions, please visit the original article at https://forum.odroid.com/viewtopic.php?f=194&t=38045.

Amiga Emulation Box: Turn Your ODROID-XU4 Into a 1980s Wonder Computer

April 3, 2020 By Brian A. Ree Linux

Things You'll Need

- An ODROID-XU4, either with a fan or heatsink
- An ODROID-XU4 case
- Either an eMMC chip or micro SD card
- A microSD card writer or eMMC writer
- Wifi Module; Module 0, 5A, and 4 are ok
- USB mouse and keyboard
- HDMI display
- Internet Connection
- Amiga Forever Plus Edition
- A Windows computer

Tools Needed

- Small screwdriver set

Introduction and Tutorial Goals

This tutorial will show you how to set up an Amiga emulator on your ODROID-XU4 by building your own copy of Uae4Arm and setting it up with kickstart ROMs and Workbench files from Amiga Forever or wherever you got your copy from. This tutorial assumes you have a working ODROID-XU4 running Ubuntu 18.XX.

For help on writing an OS image to an SD card or eMMC module please visit the links below. Make sure you select the ODROID-XU4 tab when getting an Ubuntu image.

https://wiki.odroid.com/odroid-xu4/getting_started/os_installation_guide?redirect=1#downloads Next, you will want to go ahead and assemble the ODROID-XU4 case using the directions found at http://middlemind.net/tutorials/odroid_go/mr3_build.html#part3. If you read each section listed above that should get you all setup to begin working on

Amiga emulation on your ODROID-XU4 device. The general cost of this project is around $140 depending on the configuration choices you make. Obviously if you choose to use micro SD cards and don't need a WIFI USB adapter you're going to spend less. This tutorial works with Amiga files that come with Amiga Forever's plus package which costs around $30. I highly recommend buying this package with all the kickstart ROMs and Workbench files you'll need to emulate Amiga computers on just about any emulation software, WinUAE, FS-UAE, Uae4Arm, Uae4All, etc. You will need to have access to a Windows computer to install Amiga Forever and grab a copy of the necessary files. After installing the software you can locate the Amiga OS files we need from the following directory. You can also use your own Amiga OS files; just follow along and substitute the proper files, where needed.

```
C:UsersPublicDocumentsAmiga FilesShared
```

Make a copy of the following folder on a USB key or one of your SD cards by using the micro SD to USB adapter

```
df
hdf

om - (Rename to kickstarts)
dirSystem
```

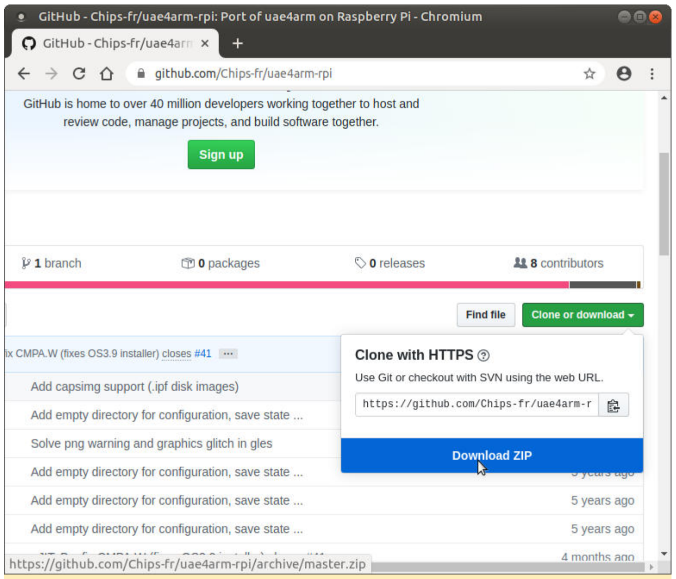
For those using their own Amiga OS files you'll need an A1200 3.1 Kickstart ROM, and a workbench 3.1 .adf or .hdf file. You'll also want to create the following folders that will be populated with files in just a bit.

```
dirGames
dirExtras
```

Now that you have all the files you need, let's get started setting up our ODROID-XU4 for Amiga emulation and Amiga games.

Compiling Uae4Arm

First, grab a copy of the latest version of Uae4Arm from the following URL: https://github.com/Chips-fr/uae4arm-rpi. Click the download zip option as shown below.


GitHub - Chips-fr/uae4arm-rpi: Port of uae4arm on Raspberry Pi - Chromium

github.com/Chips-fr/uae4arm-rpi

GitHub is home to over 40 million developers working together to host and review code, manage projects, and build software together.

1 branch 0 packages 0 releases 8 contributors

ix CMPA.W (fixes OS3.9 installer) closes #41

Add capsimg support (.ipf disk images)

Add empty directory for configuration, save state ...

Solve png warning and graphics glitch in gles

Add empty directory for configuration, save state ...

Add empty directory for configuration, save state ... 5 years ago

Add empty directory for configuration, save state ... 5 years ago

4 months ago

Clone with HTTPS

Use Git or checkout with SVN using the web URL.

https://github.com/Chips-fr/uae4arm-r

https://github.com/Chips-fr/uae4arm-rpi/archive/master.zip

Figure 1 - Downloading the zip file for uae4arm

Create two new folders in your home directory, /home/odroid. FYI, the super user password is odroid by default on the ODROID-XU4 OS images we're using. We'll want to create a install_zips folder and a uae4arm folder. When the uae4arm source code finishes downloading, copy the resulting uae4arm-rpi-master.zip file to the install_zips folder. Expand it and move the resulting uae4arm-rpi-master folder to the newly created uae4arm folder.

We'll need to install some packages before we can compile the uae4arm code. Open a new terminal by going to Menu -> System Tools -> MATE Terminal. Run the following commands in the terminal window.

```
$ sudo apt-get install libsdl1.2-dev -y
$ sudo apt-get install libguichan-dev -y
$ sudo apt-get install libsdl-ttf2.0-dev -y
$ sudo apt-get install libsdl-gfx1.2-dev -y
$ sudo apt-get install libxml2-dev -y
$ sudo apt-get install libflac-dev -y
$ sudo apt-get install libmpg123-dev -y
$ sudo apt-get install libmpeg2-4-dev -y
$ sudo apt-get install autoconf -y
$ sudo apt-get install libgles2-mesa-dev -y
$ sudo apt-get install libgles1 -y
$ sudo apt-get install libgles2 -y
$ sudo apt-get install libglvnd-dev -y
```

Or if you want to get it all done in one big command…

```
$ sudo apt-get install libsdl1.2-dev -y;sudo apt-
get install libguichan-dev -y;sudo apt-get install
libsdl-ttf2.0-dev -y; sudo apt-get install libsdl-
gfx1.2-dev -y;sudo apt-get install libxml2-dev -
```

```
y;sudo apt-get install libflac-dev -y; sudo apt-
get install libmpg123-dev -y;sudo apt-get install
libmpeg2-4-dev -y;sudo apt-get install autoconf -
y; sudo apt install libgles2-mesa-dev -y;sudo apt
install libgles1 -y;sudo apt install libgles2 -
y;sudo apt-get install libglvnd-dev -y;
```

Navigate to the uae4arm-rpi-master directory using the following command.

```
$ cd /home/odroid/uae4arm/uae4arm-rpi-master
```

Next we're going to compile uae4arm by running the following command in the terminal with the current directory set to the uae4arm-rpi-master directory.

```
$ make PLATFORM=gles
```

Allow the compilation to run its course and when it completes you should have a new file in the uae4arm-rpi-master directory named uae4arm. Copy the adf, hdf, dir, and kickstarts folders mentioned earlier from your USB drive or whatever you're using, to the uae4arm folder. You should have the following folder structure. If you aren't using Amiga Forever create the following directory structure and put your .adf files, games, Workbench, etc in the adf folder. Similarly place any .hdf files, drive images, etc, into the hdf folder. Your kickstart ROM files go into the kickstarts folder. The Games folder will be used for hard drive games and WHDLoad prepped games. Extras will be used for Amiga programs and files we'll need to transfer into the boot .hdf drive. System is used specifically for bootable .hdf files.

- uae4arm - uae4arm-rpi-master - adf - hdf - kickstarts - dir - System - Extras - Games

You should have the following folder setup in your odroid home directory and the following folders in the uae4arm directory.

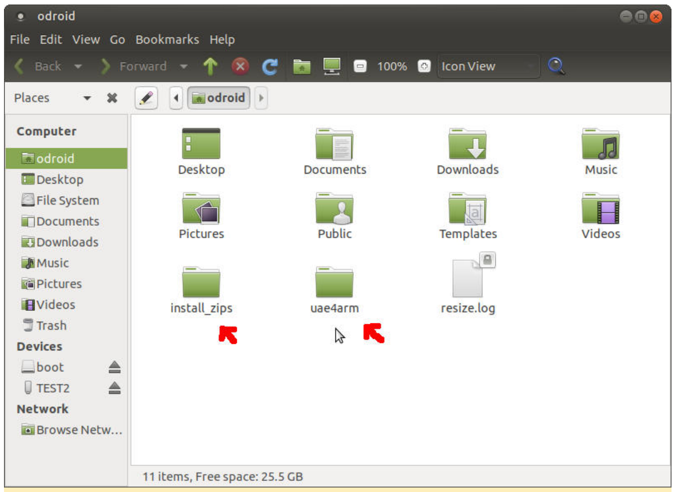
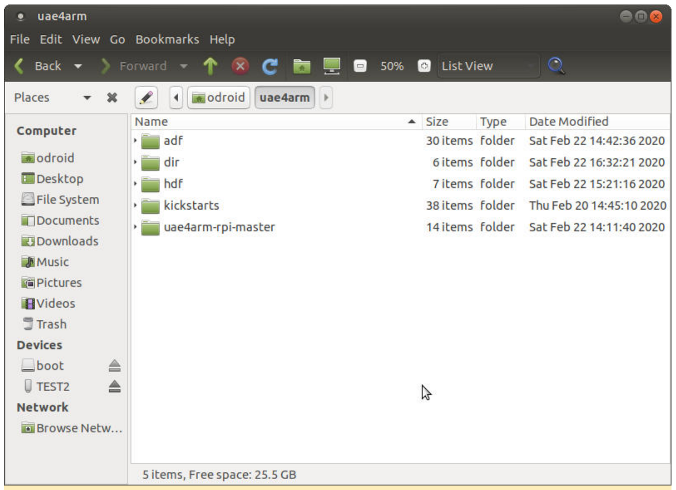
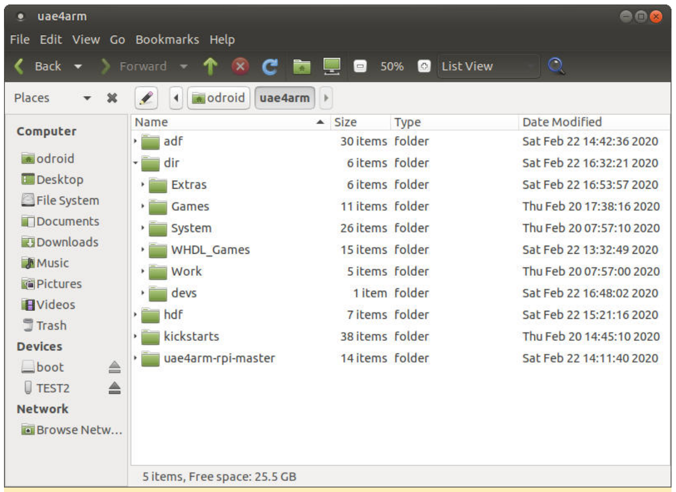


Figure 2 - You should have two folders 'install_zips' and 'uae4arm'



Name	Size	Type	Date Modified
adf	30 items	folder	Sat Feb 22 14:42:36 2020
dir	6 items	folder	Sat Feb 22 16:32:21 2020
hdf	7 items	folder	Sat Feb 22 15:21:16 2020
kickstarts	38 items	folder	Thu Feb 20 14:45:10 2020
uae4arm-rpi-master	14 items	folder	Sat Feb 22 14:11:40 2020

Figure 3 - The content of the 'uae4arm' folder

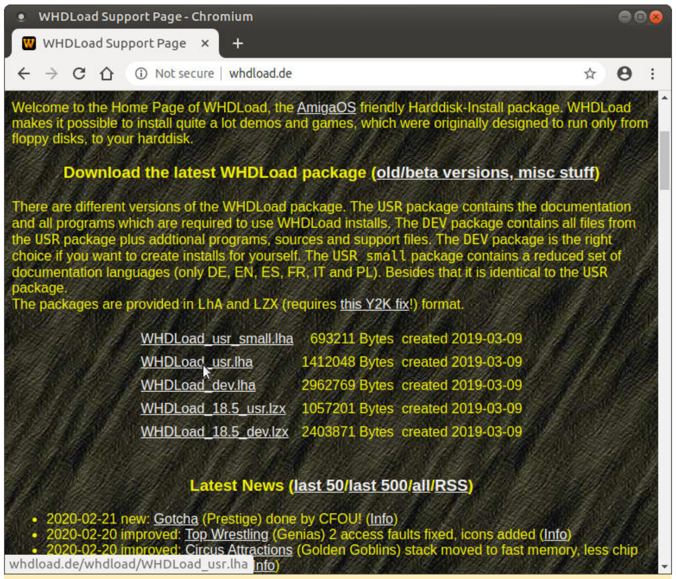


Name	Size	Type	Date Modified
adf	30 items	folder	Sat Feb 22 14:42:36 2020
dir	6 items	folder	Sat Feb 22 16:32:21 2020
Extras	6 items	folder	Sat Feb 22 16:53:57 2020
Games	11 items	folder	Thu Feb 20 17:38:16 2020
System	26 items	folder	Thu Feb 20 07:57:10 2020
WHDL_Games	15 items	folder	Sat Feb 22 13:32:49 2020
Work	5 items	folder	Thu Feb 20 07:57:00 2020
devs	1 item	folder	Sat Feb 22 16:48:02 2020
hdf	7 items	folder	Sat Feb 22 15:21:16 2020
kickstarts	38 items	folder	Thu Feb 20 14:45:10 2020
uae4arm-rpi-master	14 items	folder	Sat Feb 22 14:11:40 2020

Figure 4 - The content of the 'dir' folder

We'll need a few pieces of software for some of the Amiga OS configuration steps I'm going to show you. You can download WHDLoad at this URL: http://whdload.de/. Download the WHDLoad_usr.lha file from the main page. When the download completes, move the file from the Downloads folder to the uae4arm/dir/Extras folder.

The WHDLoad site is shown below.



Welcome to the Home Page of WHDLoad, the AmigaOS friendly Harddisk-Install package. WHDLoad makes it possible to install quite a lot demos and games, which were originally designed to run only from floppy disks, to your harddisk.

Download the latest WHDLoad package (old/beta versions, misc stuff)

There are different versions of the WHDLoad package. The USR package contains the documentation and all programs which are required to use WHDLoad installs. The DEV package contains all files from the USR package plus addtional programs, sources and support files. The DEV package is the right choice if you want to create installs for yourself. The USR `small` package contains a reduced set of documentation languages (only DE, EN, ES, FR, IT and PL). Besides that it is identical to the USR package.
The packages are provided in LhA and LZX (requires this Y2K fix!) format.

WHDLoad_usr_small.lha	693211 Bytes	created 2019-03-09
WHDLoad_usr.lha	1412048 Bytes	created 2019-03-09
WHDLoad_dev.lha	2962769 Bytes	created 2019-03-09
WHDLoad_18.5_usr.lzx	1057201 Bytes	created 2019-03-09
WHDLoad_18.5_dev.lzx	2403871 Bytes	created 2019-03-09

Latest News (last 50/last 500/all/RSS)

- 2020-02-21 new: Gotcha (Prestige) done by CFOU! (Info)
- 2020-02-20 improved: Top Wrestling (Genias) 2 access faults fixed, icons added (Info)
- 2020-02-20 improved: Circus Attractions (Golden Goblins) stack moved to fast memory, less chip

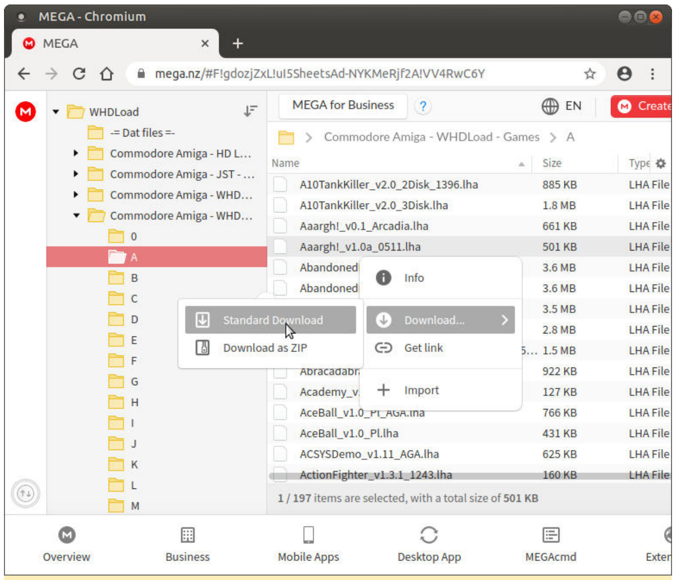
whdload.de/whdload/WHDLoad_usr.lha nfo)

Figure 5 - The WHDLoad website

Next we'll need an Amiga decompression tool called lha. You can download a copy directly at the following URL: lha.run. When the file download is complete move the resulting file into the uae4arm/dir/Extras folder.

The last thing we'll need are some games to test. Most of the Amiga software library is considered abandon-ware since it's a dead platform. For this tutorial, you'll need to find some .adf games for Amiga 500, 1000, or 1200. You'll also need to download some games prepped for the WHDLoad software from this URL WHDLoad Games. The games must come from the WHDLoad directory.

The game download site is as shown below.



Name	Size	Type
A10TankKiller_v2.0_2Disk_1396.lha	885 KB	LHA File
A10TankKiller_v2.0_3Disk.lha	1.8 MB	LHA File
Aaargh!_v0.1_Arcadia.lha	661 KB	LHA File
Aaargh!_v1.0a_0511.lha	501 KB	LHA File
Abandoned	3.6 MB	LHA File
Abandoned	3.6 MB	LHA File
	3.5 MB	LHA File
	2.8 MB	LHA File
5...	1.5 MB	LHA File
Abracadabr	922 KB	LHA File
Academy_v	127 KB	LHA File
AceBall_v1.0_Pl_AGA.lha	766 KB	LHA File
AceBall_v1.0_Pl.lha	431 KB	LHA File
ACSYSDemo_v1.11_AGA.lha	625 KB	LHA File
ActionFighter_v1.3.1_1243.lha	160 KB	LHA File

1 / 197 items are selected, with a total size of 501 KB

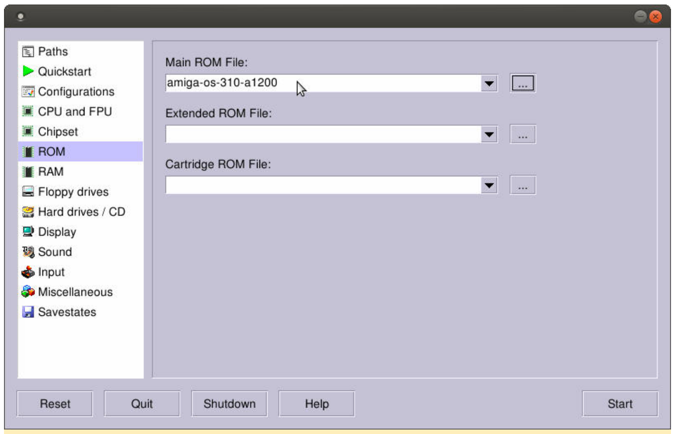
Figure 6 - Download site website

What WHDLoad does is it launches Amiga games from the hard drive and not from a disk image so you can have a whole collection loaded on your emulated Amiga OS. It also soft kicks the game's required kickstart ROM version. This way you won't have to load a new disk every time you want to play a game.

Go back to the terminal window or open up a new one. Navigate to the uae4arm-rpi-master folder, cd /home/odroid/uae4arm/uae4arm-rpi-master. Next start uae4arm by running ./uae4arm. This will execute the binary file you compiled earlier. If you get an error that the file cannot be executed run the following commands in the terminal.

```
sudo chmod 755 ./uae4arm
sudo chmod +x ./uae4arm
```

Run the ./uae4arm command again and you should see the Uae4Arm launcher window popup as shown below.

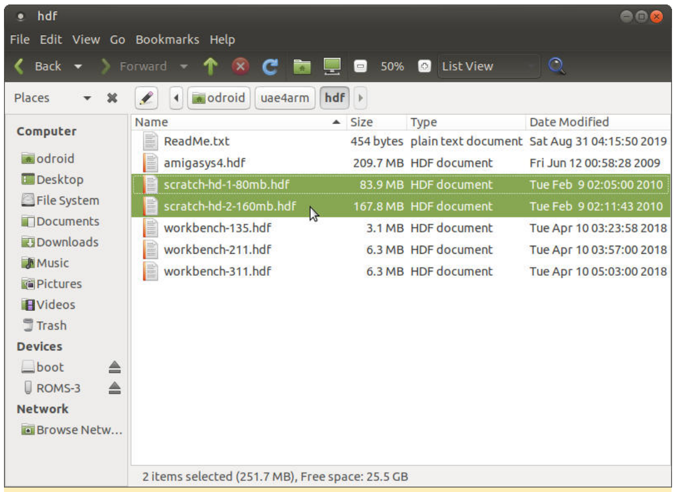


Main ROM File: amiga-os-310-a1200
Extended ROM File:
Cartridge ROM File:

Figure 7 - ROM Selection

Configured Uae4Arm

In this section, we'll work on configuring Uae4Arm so that we have a stable Amiga emulator running which we can use to launch a bunch of games. Before we get into the details, we're going to need two more files. The create hard drive file feature of Uae4Arm seems broken, so we'll have to work around it. Go to this URL: https://scruss.com/blog/2010/02/07/amiga-blank-hdf-images/ and download both the 80MB and 160MB blank hard drive files.



Name	Size	Type	Date Modified
ReadMe.txt	454 bytes	plain text document	Sat Aug 31 04:15:50 2019
amigasys4.hdf	209.7 MB	HDF document	Fri Jun 12 00:58:28 2009
scratch-hd-1-80mb.hdf	83.9 MB	HDF document	Tue Feb 9 02:05:00 2010
scratch-hd-2-160mb.hdf	167.8 MB	HDF document	Tue Feb 9 02:11:43 2010
workbench-135.hdf	3.1 MB	HDF document	Tue Apr 10 03:23:58 2018
workbench-211.hdf	6.3 MB	HDF document	Tue Apr 10 03:57:00 2018
workbench-311.hdf	6.3 MB	HDF document	Tue Apr 10 05:03:00 2018

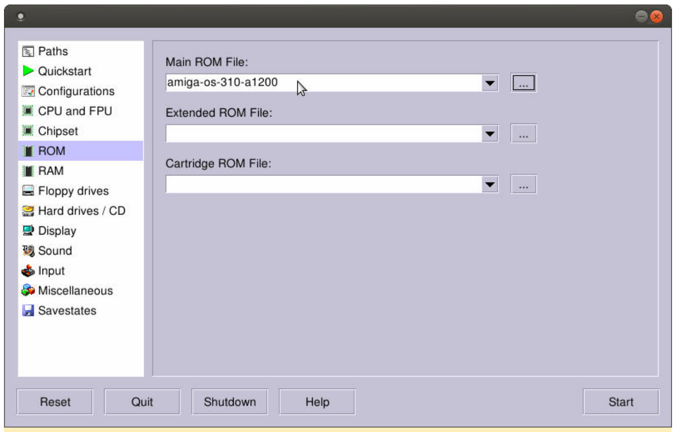
2 items selected (251.7 MB), Free space: 25.5 GB

Figure 8 - hdf files

When the files are done downloading, move them from the Downloads folder to the install_zips folder. Expand the files. When this is done move the resulting files to the hdf folder under the uae4arm directory.

Now we'll set the proper configuration to create a nice A1200 emulated Amiga system. Go to the Quick Start configuration section and select A1200 from the list. Next go to the CPU and FPU section. Make sure JIT is selected under both CPU and FPU sections. Select the Fastest option for the CPU speed and set the FPU to 68882.

Next, go to the ROM section and load the amiga-os-3101-a1200.rom file from the kickstarts folder. You can also use an equivalent file if you have one, the file we're using we got from: https://www.amigaforever.com/ ; this file, and many more, come with their plus offering. The screenshot below shows the ROM screen with the proper file selected.



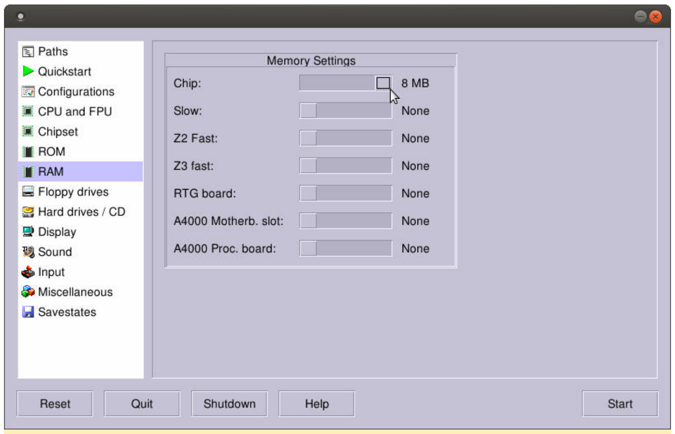
Main ROM File: amiga-os-310-a1200

Extended ROM File:

Cartridge ROM File:

Figure 9 - Rom Selection

Next we'll configure the RAM section. Go to the RAM section and slide the Chip RAM to 8MB as shown below.

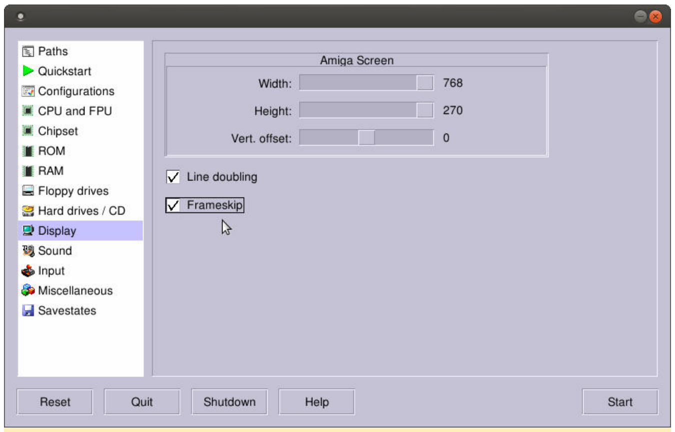


Memory Settings

Chip:	8 MB
Slow:	None
Z2 Fast:	None
Z3 fast:	None
RTG board:	None
A4000 Motherb. slot:	None
A4000 Proc. board:	None

Figure 10 - RAM Settings

Now we'll load up some hard drives. The first drive will be our boot drive. Navigate to the dir/System directory and select the workbench-311.hdf. Again this file comes from the Amiga Forever package but you can use a bootable .adf version if you have one or another bootable hard drive that has Workbench 3.1 on it. The screenshot below shows the drive configuration.



Amiga Screen

Width:	768
Height:	270
Vert. offset:	0

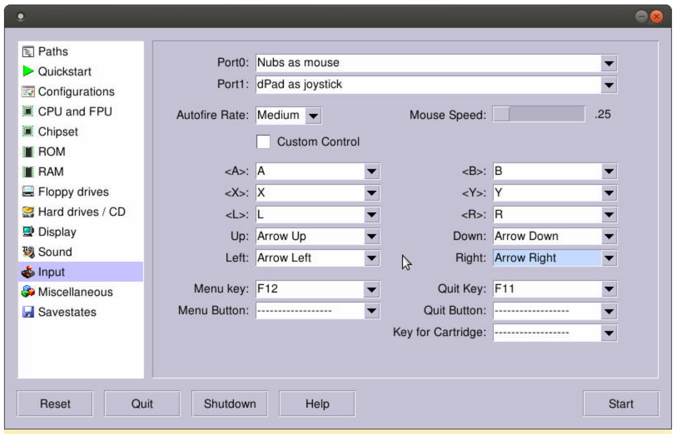
[✓] Line doubling

[✓] Frameskip

Figure 11 - Hard Drive Settings

The next hard drive we want to load up is one of the blank drives we downloaded from the Internet. I would go with the 160MB one. Load it up into the list of drives and make sure to unselect the bootable option. What I'm going to show you how to do here is copy over a smaller boot drive onto a larger blank drive then use that one as a boot drive. Uae4Arm does a really good job at emulating A1200 and earlier machines, but I can't get the A4000 entries to work. The software is at version 0.5 and might need some work. One other quirk is that it can't write new files or folders to directory based drives; there is a mode error that crashes the application. A workaround is to use a hdf based drive. So our goal is to create a new larger boot drive so we can move around some games and install some new software.

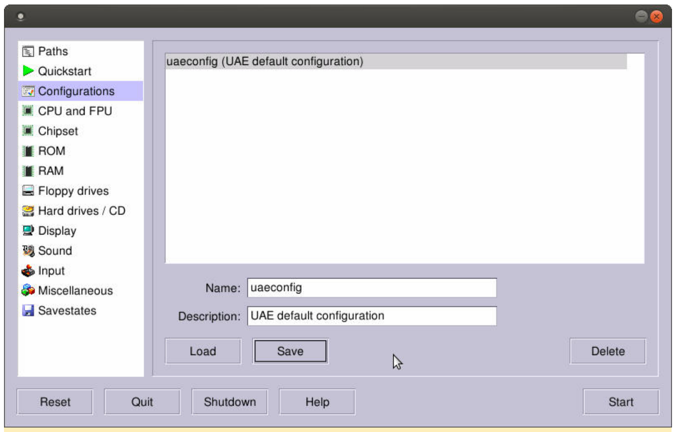
Let's configure the screen size and set the emulator to use the largest possible size under the Display configuration section. The screenshot below shows the largest possible screen size with an 8MB A1200 and no graphics card. Next, let's set up the input. Go to the Input configuration section. Setup the input how you want, I used the following configuration to start since all the mappings go directly to keys with the same name.



Port0: Nubs as mouse
Port1: dPad as joystick
Autofire Rate: Medium
Mouse Speed: .25
Custom Control
<A>: A
<B>: B
<X>: X
<Y>: Y
<L>: L
<R>: R
Up: Arrow Up
Down: Arrow Down
Left: Arrow Left
Right: Arrow Right
Menu key: F12
Quit Key: F11
Menu Button: -----------------
Quit Button: -----------------
Key for Cartridge: -----------------

Figure 12 - Input Settings

One last thing to do before we fire up our Amiga A1200 is go back to the Configurations section and click the Save button. You can change the configuration name but I just use the default.



uaeconfig (UAE default configuration)
Name: uaeconfig
Description: UAE default configuration

Figure 13 - Configurations Settings

Now that you have everything all configured and ready to go click the Start button and you should see a nice clean Amiga A1200 Workbench loaded up as depicted below.

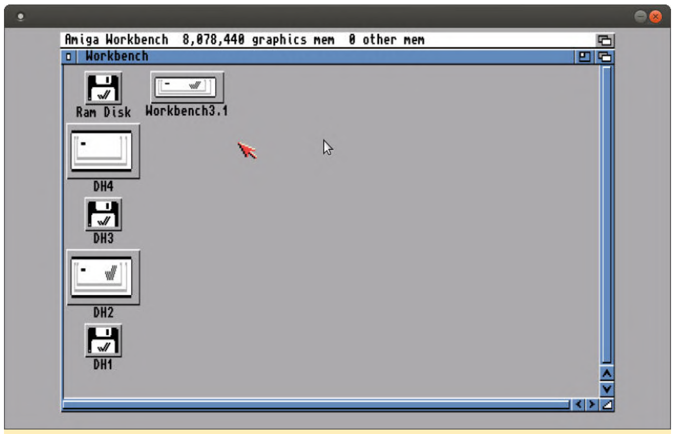
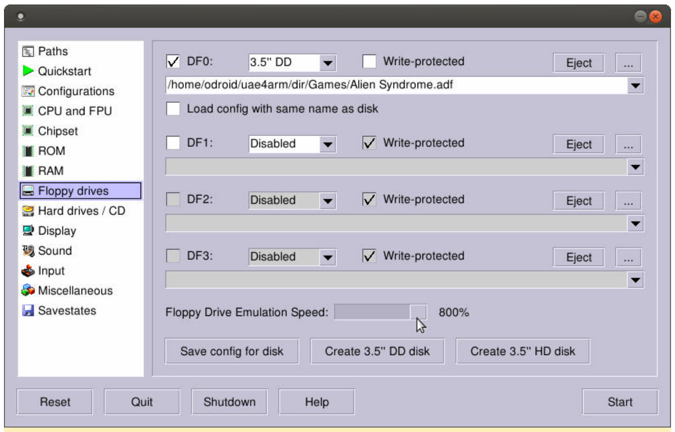


Figure 14 - Amiga Workbench

Running *.adf Format Amiga Games

Now the A1200 can run, for the most part, A500, A600, and A1000 games. However, not every one of them will work. Some may require a specific chipset, but most should work "out-of-the-box." To load up and run an .adf format game press the F12 key, or whichever key you mapped in the Input configuration section.

Go to the Floppy drives configuration section and select an .adf game you want to play. You can even check the Load config with the same name as disk option to create specific configurations for the given disk/game. For now, leave that option unchecked. Select your game disk as shown below.



✓ DF0: 3.5" DD ☐ Write-protected
/home/odroid/uae4arm/dir/Games/Alien Syndrome.adf
☐ Load config with same name as disk
☐ DF1: Disabled ✓ Write-protected
☐ DF2: Disabled ✓ Write-protected
☐ DF3: Disabled ✓ Write-protected
Floppy Drive Emulation Speed: 800%

Figure 15 - Floppy Disk Settings

Click the Resume button if you already started your A1200 emulation or click the Start button. You should see a floppy disk icon on the Workbench desktop. Double click it to open it. Hold down the right mouse button while your mouse is at the workbench header, hover over Window, then Show, and select All Files.

Now you should be able to see all files on the game floppy disk you loaded. Find the game executable and double click it. You should now be able to play the selected game on your ODROID-XU4 powered Amiga A1200 emulator. If the game crashes you may need to look up specific hardware requirements for that game and adjust your emulator's configuration to meet those needs.

The following screenshot shows the crack screen of a game I've loaded up from a floppy disk image, .adf file.

```
ALIEN SYNDROME - MEGA TRAINER
ANOTHER MASTERPIECE OF THE LIGHTFORCE

USE MOUSE TO SELECT

UNLIMITED LIVES      NO
UNLIMITED TIME       NO
STARTING ROUND        1
WEAPONRY              S
PLAYER COLLISION     ON
---- START   GAME ----
```

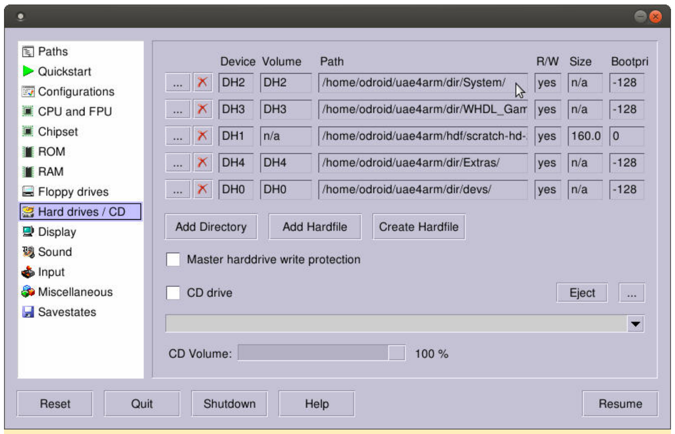
Figure 16 - Alien Syndrome

Using WHDLoad to Run Games

In this section we'll cover how to run Amiga games from the hard drive using WHDLoad. This approach has the added advantage of using a soft kick to launch the game with the proper kickstart ROM. You'll need to have a workbench boot drive and an empty harddrive, .hdf, that we downloaded earlier. I'm not sure if this process will work with a bootable workbench floppy disk, it might. I just haven't tried it.

So I used the Workbench-311.hdf from my copy of Amiga Forever but it's only about a 9MB drive. You'll run out of space very quickly. To create a new bootable drive with more free space, I've loaded up the 160MB blank .hdf drive, and we're going to clone the boot drive onto it.

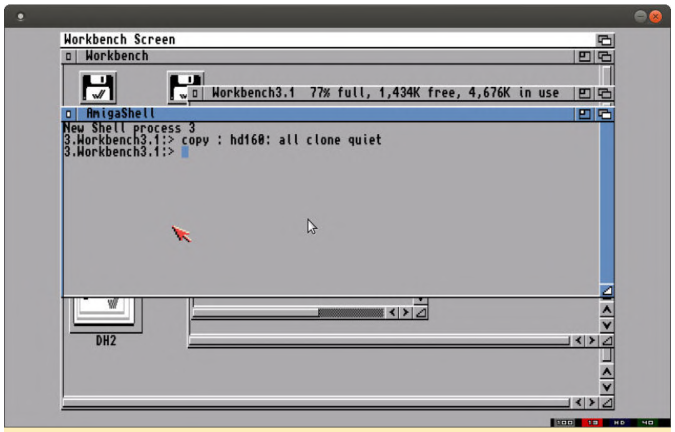
My initial harddrive setup is shown below. The key things to remember are that we only use "directory" based harddrives as a source for files. We won't be creating any new files or folders on them. So the Extras folder is where we'll store new Amiga software, DH4. The first drive is a smaller, but, preconfigured boot drive, DH2. Lastly the new blank 160MB harddrive is going to be our new boot drive and will eventually hold all the new Amiga software we need, DH3.



Device	Volume	Path	R/W	Size	Bootpri
DH2	DH2	/home/odroid/uae4arm/dir/System/	yes	n/a	-128
DH3	DH3	/home/odroid/uae4arm/dir/WHDL_Gam	yes	n/a	-128
DH1	n/a	/home/odroid/uae4arm/hdf/scratch-hd-	yes	160.0	0
DH4	DH4	/home/odroid/uae4arm/dir/Extras/	yes	n/a	-128
DH0	DH0	/home/odroid/uae4arm/dir/devs/	yes	n/a	-128

Figure 17 - Hard Drive Settings

Find the AmigaSHELL program, it should be in the Workbench 3.1 drive's System folder. Double click the Shell icon. And run the following command.



```
New Shell process 3
3.Workbench3.1:> copy : hd160: all clone quiet
3.Workbench3.1:>
```

Figure 18 - AmigaShell

The AmigaSHELL command is copy : hd160: all clone quiet, where the second drive name, hd160 is whatever name your 160MB drive was assigned. This command will clone the drive and create a new bootable drive. When it completes you can close your emulation session and then remove the smaller drive and map the larger 160MB drive as the boot drive.

You're also going to map a new drive based on the Extras directory. This directory should contain the lha.run, and WHDLoad.lha files. You should also map the Games directory that has the WHDLoad prepped games. So you'll have a total of three drives listed below.

1. Larger boot drive, 160MB, based on .hdf 2. Extras drive, based on directory dirExtras 3. Games drive, based on directory dirGames

Since I've had trouble creating new files and folders in a directory based drive, not sure what was the cause. I always copy the programs I need to run to the boot .hdf drive. So in this case drag and drop the lha.run, WHDLoad.lha, and target game .lha files to the boot drive.

If you can't see the files in any drive or folder you're viewing simply right-click in the Workbench 3.1 header and while holding the button down drag the cursor to Window, then Show, and select All Files.

Once the files have copied over, run the lha.run program by double clicking on it. Make sure you're using the copy that's on the boot drive. Once it has completed installing, open up a new AmigaSHELL and type lha and press enter. You should see a list of arguments for the command.

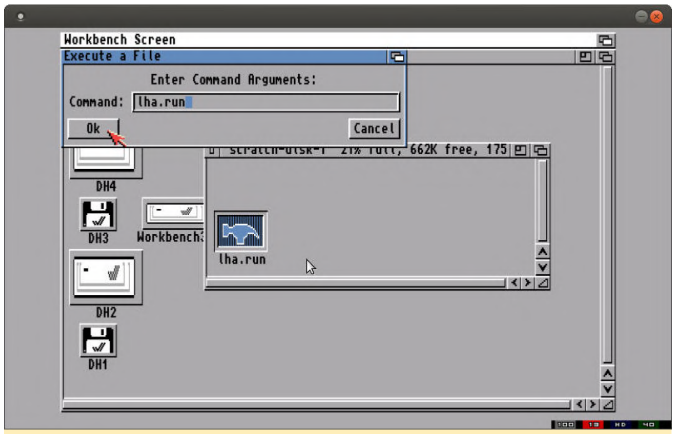
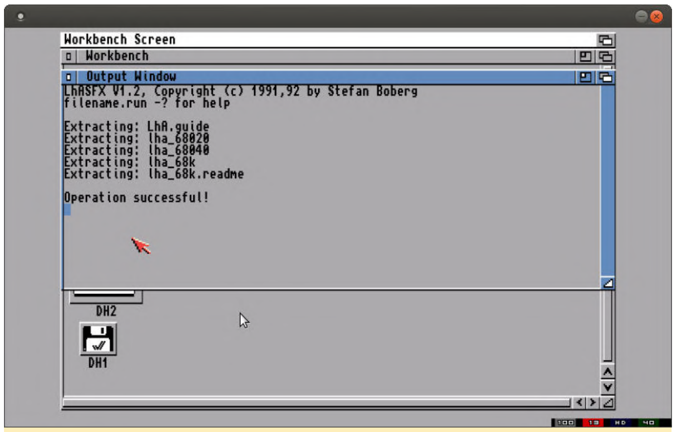


Figure 19 - Execute a file

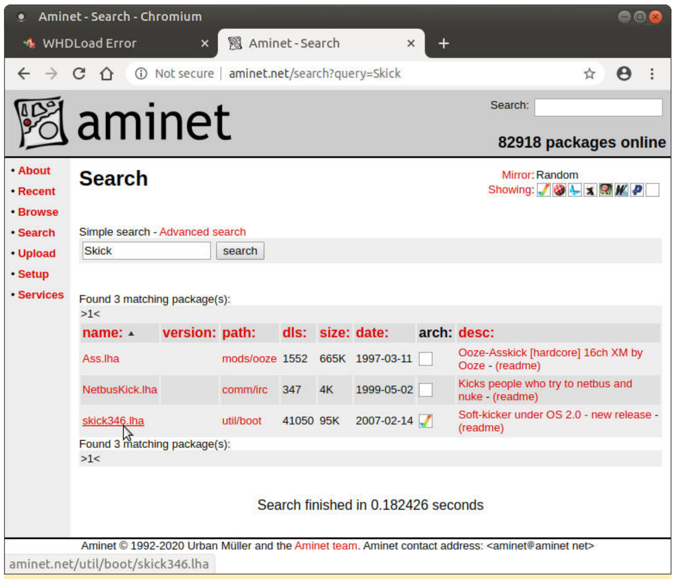


```
LhASFX V1.2, Copyright (c) 1991,92 by Stefan Boberg
filename.run -? for help

Extracting: LhA.guide
Extracting: lha_68020
Extracting: lha_68040
Extracting: lha_68k
Extracting: lha_68k.readme

Operation successful!
```

Figure 20 - Output Window

You should be on the root of the boot drive and that's where WHDLoad should have been copied. From the AmigaSHELL, run lha x WHDLoad.lha. Let the expansion run and when it's complete find the WHDLoad folder on the boot drive. Open it and run the Install program. You can choose options to run the install as a test and not actually install it. That's fine to do first to make sure everything runs smoothly. Complete a real install and you're almost ready to run WHDLoad prepped games. I'm not going to cover how to convert .adf games to WHDLoad games in this tutorial. Using the links above will get you a whole bunch of games prepped for WHDLoad.

We're going to need to download one more piece of software before we can launch a WHDLoad game, however. Navigate your browser to: http://aminet.net and search for skick. Download a copy of the program as shown as follows.



Search

Simple search - Advanced search

Skick | search

Found 3 matching package(s):
>1<

name: ▲	version:	path:	dls:	size:	date:	arch:	desc:
Ass.lha		mods/ooze	1552	665K	1997-03-11		Ooze-Asskick [hardcore] 16ch XM by Ooze - (readme)
NetbusKick.lha		comm/irc	347	4K	1999-05-02		Kicks people who try to netbus and nuke - (readme)
skick346.lha		util/boot	41050	95K	2007-02-14	✓	Soft-kicker under OS 2.0 - new release - (readme)

Found 3 matching package(s):
>1<

Search finished in 0.182426 seconds

Aminet © 1992-2020 Urban Müller and the Aminet team. Aminet contact address: <aminet@aminet net>

Figure 21 - The aminet.net website

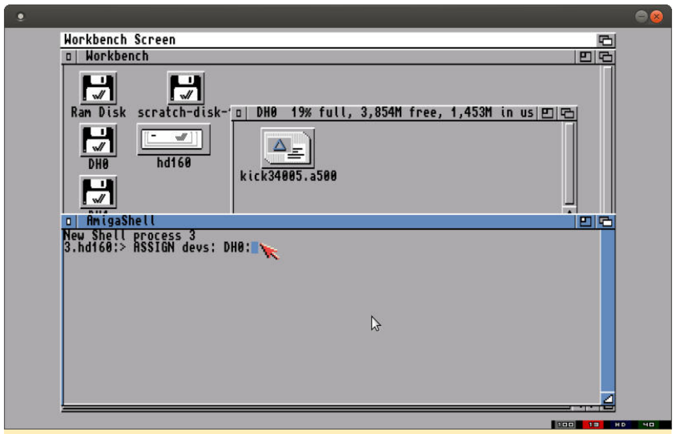
Shutdown your Amiga A1200 emulator and copy the new skick346.lha file into the Extras directory with the other Amiga programs. Start up your A1200 and copy the skick346.lha file to the boot drive. Open up a shell, Workbench 3.1 -> System -> Shell. Create a new folder on the boot drive named kickstarts. Expand the file by running this command in the shell, lha x skick346.lha :kickstarts. Use the full name of the skick file you've downloaded if you have a different version.

Also, while you have the shell open, expand the WHDLoad prepped game by running lha x name_of_game.lha. Now you should have new folders on the boot drive. Run the WHDLoad prepped game file and you'll get some errors that look like the following.

```
DOS-Error #205 (object not found) on reading
"devs:kickstarts/kick40068.a4000
```

Make a copy of the a4000, or a500, or whatever missing kickstart file is indicated and move it into the boot drive in the same way you've moved the other Amiga programs. Copy it into the kickstarts folder on the boot drive. Make sure that it has been renamed to match the error. In my case the error was a missing a kick34005.a500 file so I used a 3.X kickstart ROM from the Amiga Forever set of included ROMs. For more information on which kickstart ROM to use with regard to Amiga Forever go to this URL: http://www.whdload.de/docs/en/need.html.

You'll need to create a symbolic link so that the devs: drive exists. Open up AmigaSHELL and run the following command, ASSIGN devs: hd0:, where the drive named hd0 is the name of your boot drive or the location of the kickstarts folder. Now you should have the path devs:kickstarts mapped, and no longer get the missing kickstart error that was shown above.



```
New Shell process 3
3.hd160:> ASSIGN devs: DH0:
```

Figure 22 - Assigning devs

Once you get past that error, run the WHDLoad prepped game again. If you get an error about a missing RTB file then double check that you decompressed the skick346.lha archive into the proper location. If you still get a kickstart ROM error then double check that you have copied and properly renamed the correct kickstart ROM file. Also, make sure that you have used the ASSIGN command correctly so that you have a devs:kickstarts drive and folder that contains RTB files and kickstart ROMs.

The Devs folder should look like what's shown below. It now contains the kick ROM you copied there and all the RTB configuration files from the soft kick Amiga program.

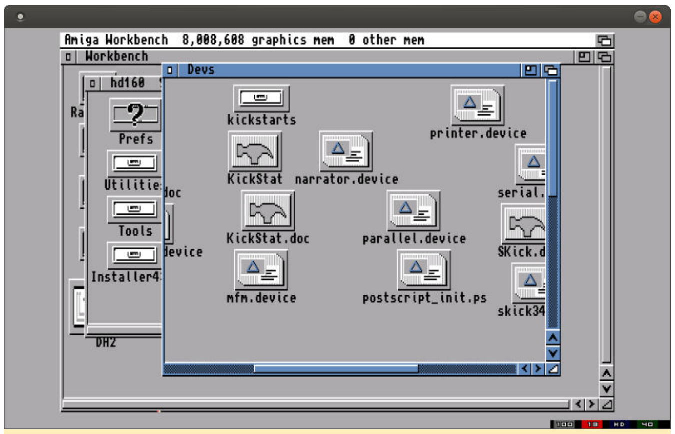


Figure 23 - Devs

I used a helicopter game that was WHDLoad prepped and completed the above mentioned steps. The screenshot below shows the game up and running. Loaded cleanly from the hard drive giving us the ability to run a bunch of games quickly and easily right from the emulated Amiga system, no floppy drive swapping.

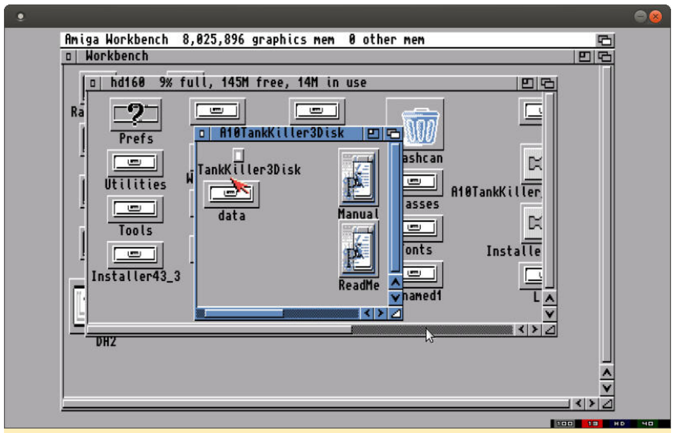


Figure 24 - Launching tankKiller3D

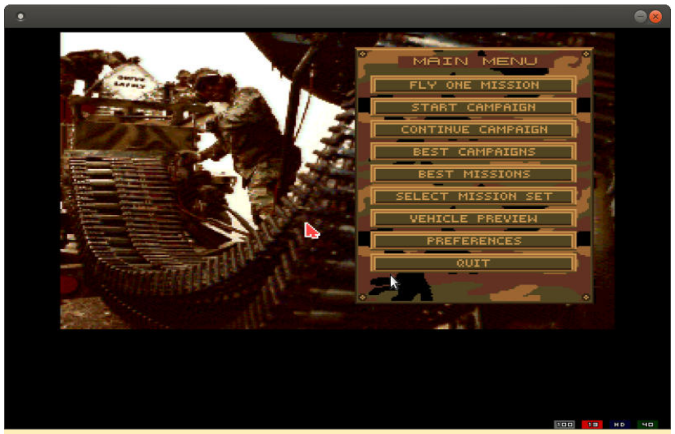


Figure 25 - Time for some fun

For more information, please visit the original article at http://middlemind.net/tutorials/odroid_go/mab_build.html.

BMW oDrive Car PC

April 1, 2020 By @FntX Tutorial

I have been working on this project for almost a year now and feel confident enough now to present it to the public - the almighty iDrive supplemental solution for all people who do not have interest in fitting a 1500€ retrofit solution.

The Project consists of the following:

- Odroid N2 4GB with 32GB eMMC and Android as the car-pc itself
- Odroid VU7a+ Screen
- Screen casing assembly to be fitted as a replacement for the small glove box on the dashboard (See: https://www.ebay.de/itm/Monitorhalter-f ... 1438.l2649)
- iDrive 7-Button Controller (Facelift)
- Matching center console assembly for the iDrive Controller
- Custom command & control board with: Arduino Nano 33 IoT, RN42HID Bluetooth Module, MCP2515 + TJA1050 CAN interface, INA219 voltage sensor, 2 pwm headers for fans (See: https://github.com/Neuroquila-n8fall/od ... rpc-nano33)
- Teltonika RUT850 automotive 4G access point
- LTC3780 power supply module
- A few wires to connect everything
- A little bit of tinkering

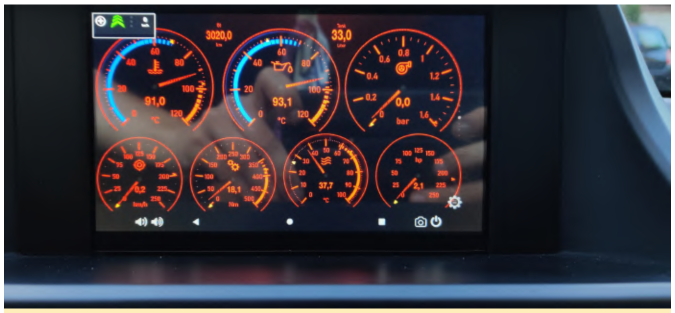


Figure 1 - Fitted VU7a+ Display on the dashboard running Torque App

Note the pin header on the power button side (left hand side). I have soldered these on to have a proper connection for triggering the power button remotely.

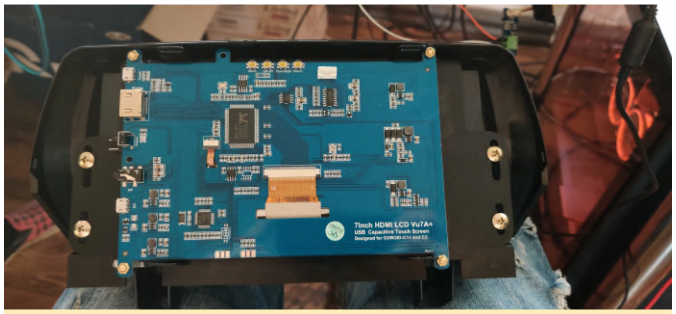


Figure 2 - VU7a+ mounted inside dash casing

At the heart of the project is the Odroid N2 itself. It resembles the computer which can be found on the iDrive system as well. It does not do much, however, because it is either on or off and any change to the power state would mean pushing a button of some sort. It works on its own but to get an integrated solution the N2 has to be powered on or off depending on the current state of the car and "drivers demand". Also it has no integrated 4G connectivity which makes it, at first sight, useless as an infotainment system with navigation and streaming services.

Since there are harsh conditions inside a car, it has to be supplied by a power supply which not only delivers enough power but also has a high input tolerance and temperature range. At this point I am very surprised the N2 itself holds together quite nicely under these conditions!

Startup and Shutdown in a Car

We could trigger the power by checking components that will come online if the car is opened or the ignition is turned on. This would mean that as soon as we launch the engine, the PC would start. This would work but is not the same as we know it from the original iDrive system. It boots very fast and we simply cannot achieve these speeds if we want to keep the android system as it is.

So I decided to hack into the CAN Bus of the car and check if I can get any messages that would tell what the current state of the car is and what is expected. This way we could start the N2 as soon as we push the "open" or "close" button on the key fob. The time it takes to walk up to the car the N2 has enough time to boot up and as soon as we enter the car it is ready to do its job.

I have also tinkered with the possibility to have the pc always on and put it to sleep or shut it down if the battery is too low. Simply put this did not work out in the end because I swapped to a Due MCU and could not make it work with a matching voltage divider network to monitor the battery. The code still exists but now that I have all the correct facilities in place, I could re-activate the mechanisms.

Screen Brightness

A problem when driving a car is that the light conditions change continuously. This makes it hard to read a display and that is the reason why we are now slowly seeing big displays coming to cars. They are replacing entire dashboard instruments. These screens need to be glare-resistant, very bright and robust. However, that is not the whole story. The screen needs to adapt to the actual outside light levels so the display is readable at bright sunlight and does not dazzle you at night.

Usually we would rely on a photo-diode to measure light levels and react on them but in this case, since we are already on the CAN Bus of the car, we can use the light level sensor on the windscreen. This is a whole module fitted on the foot-end of the rear-view mirror called "RLS" (Regen-Licht-Sensor = Rain-Light-Sensor). It reports the current light levels at intervals or when it changes. We can use that to feed in a PWM signal to the VU7a(+) backlight regulator as described on the Wiki (https://bit.ly/2UAzK1z).

Command & Control Board

The "Command & Control" module is the interface between the Car and the N2. It bridges the gap between the Android system and the Cars CAN-bus network and takes care of all the tasks that would otherwise mean a customization of the android system or writing apps. The first iteration of the control board was a mixture of a loose Arduino Due, Bluetooth module and MCP2515 CAN Interface module. What a mess!

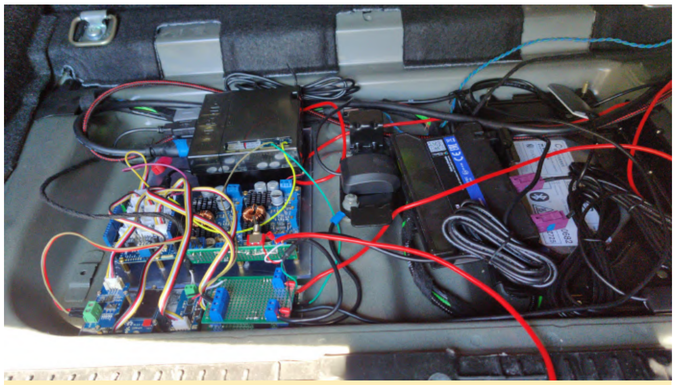


Figure 3 - This is the first iteration. It is really a complete mess but somehow I needed an evaluation platform.

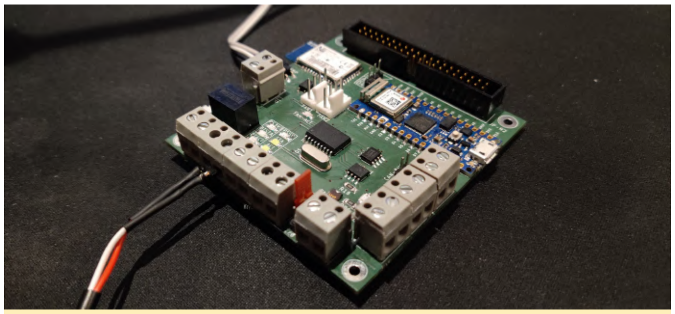
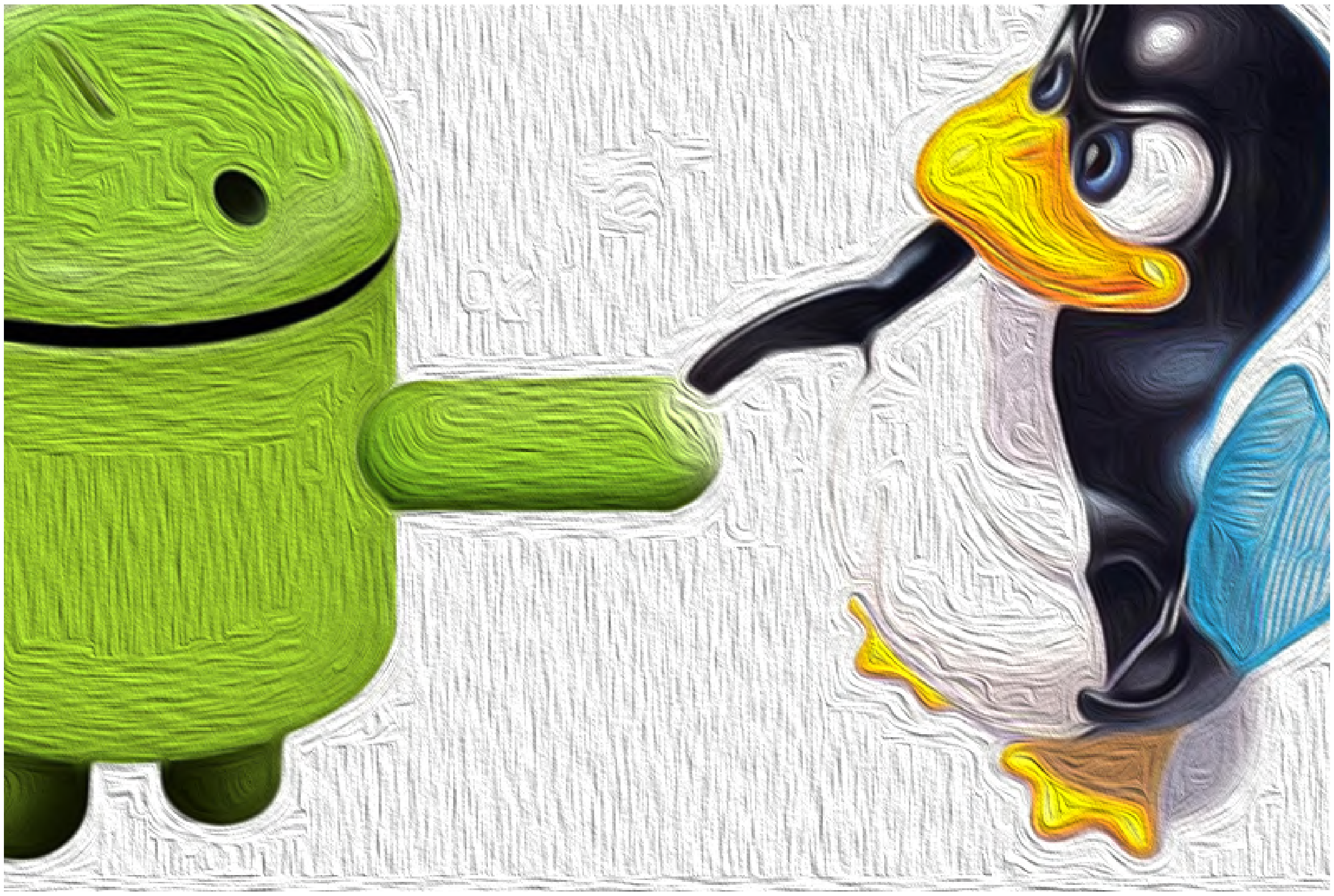


Figure 4 - One year later, the whole thing is going to be "final"

The source code and detail can be found at the Github repo: https://bit.ly/2QMx7bR. I will upload more pictures of the whole setup on the forum later, when I am finally able to swap out the cable mess with the integrated board. For more information, please visit the original post at https://forum.odroid.com/viewtopic.php?f=182&t=37944.

Running Linux Under Android

April 1, 2020 By Adrian Popa Android, Linux, Tutorial



The ODROID-N2 seems like a perfect Android TV system - everything runs smoothly and converts a regular TV into a valuable smart TV. All is well if that is the only thing you want to do with your ODROID-N2, but for me, it is not enough. I wanted to use the ODROID-N2 to replace an ODROID-C2 that was running Linux, had some sensors (temperature and PIR) attached to its GPIO pins, and also ran Kodi. I could have used Linux on the ODROID-N2, but the Android TV experience was nicer for a media center.

So I embarked on a new journey of learning how to make Android and Linux coexist on the same hardware so that I could easily transfer my linux scripts that handled the sensor data from ODROID-C2 to ODROID-N2.

If you are starting your project from scratch you should look into Hardkernel's Android Things API implementation (https://magazine.odroid.com/article/android-things/) and build native Android apps that use the GPIO pins, but in my case I already had some python scripts that read temperature and movement data so I wanted to reuse them (https://bit.ly/2xhnGdH).

Installing Linux under Android TV

In order to port existing scripts to Android It is better to install a Linux environment in a chroot. The best app for the job is Linux Deploy (https://bit.ly/2UbeKPV) which you can find on the Play Store, but unfortunately It is not available for Android TV. You will need to download the apk from a different source (I used apkpure https://bit.ly/2wuiHGq) and install it on ATV. To install an apk package you have two options:

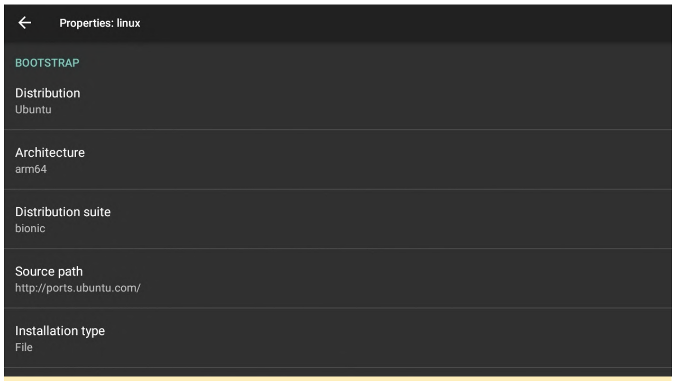
- Copy the apk file on a USB drive, attach it to the ATV and use a file explorer (there are some options available on the Play Store) to install it.
- Or install it via adb (either with a usb cable, or over the network):

```
$ adb connect odroid-ip
$ adb install ~/downloads/Linux
Deploy_v2.6.0_apkpure.com.apk
```

I am using voodik's ATV image (https://bit.ly/3bkj3OI) with adb enabled over the network.

Once Linux Deploy is installed, use a mouse/keyboard (it is not designed for ATV and remote control input does not cover all options) to set up and deploy a Linux distribution of your choice on the ODROID. If you do not have access to a suitable input device you can install scrcpy (https://github.com/Genymobile/scrcpy) and access Android in a remote-desktop-like fashion from your computer. You may also need to install an extra launcher that sees non ATV apps, like Sideload Launcher (https://play.google.com/store/apps/details?id=eu.chainfire.tv.sideloadlauncher&hl=en).

Start the app and select the properties icon and set up your environment. Note that it requires root access, so make sure to grant it root. Here you can choose your favourite distro (I used Ubuntu 18.04), you can choose the size of the rootfs image (I went with 8G) and set up default credentials.


Properties: linux

BOOTSTRAP

Distribution
Ubuntu

Architecture
arm64

Distribution suite
bionic

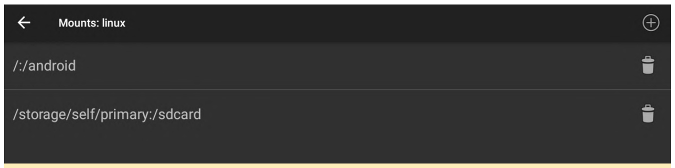
Source path
http://ports.ubuntu.com/

Installation type
File

Figure 1 - Linux deploy configuration properties

Under the INIT section make sure you Enable an initialization system. I went with sys-v which uses scripts in /etc/rc3.d/.

Under the mount options I wanted to expose some android settings to the guest Linux image, so I mapped / and /sdcard to internal mount points in Linux. Fortunately /dev and /sys are already shared.


Mounts: linux

/:/android

/storage/self/primary:/sdcard

Figure 2 - Linux deploy mount points

In theory you can install in a directory and benefit from all the host's space, but for me that option did not work.

When you are ready you can select Start and the program will start downloading the required packages and build a minimal image for you. This may take a while, but when It is done you can go ahead and start that image. If you have enabled ssh you should now be able to log in to your ODROID's IP with ssh on port 22. Now you can use apt to update the package list and install the packages you need.

In my case I needed python3 and a bunch of python modules, so I installed them with:

```
# apt-get install python3-pip python3-paho-mqtt
```

Since I need to use the GPIO pins, I wanted to install Hardkernel's wiringpi branch for the ODROID-N2. The simplest way to do it (without compiling anything) is to grab it from Hardkernel's PPA as described in the wiki at https://wiki.odroid.com/odroid-n2/application_note/gpio/wiringpi:

```
# apt install software-properties-common
# add-apt-repository ppa:hardkernel/ppa
# apt update
# apt install odroid-wiringpi odroid-wiringpi-
python
```

Once installed, verify that wiringpi can access the GPIO pins and read their state

```
# gpio readall
```

If all is well, transfer over your scripts and hook up the hardware. In my case I am using a DS18B20 temperature sensor (https://www.adafruit.com/product/381) that uses one-wire for communication and a HC-SR501 (https://www.hardkernel.com/shop/motion-detector-hc-sr501/) PIR sensor. The PIR sensor is the easiest to work with since it requires a GPIO pin set to input and you can simply read its state. I am using physical pin 12 which corresponds to wiringpi pin 1 if you consult the mapping (https://wiki.odroid.com/odroid-n2/hardware/expansion_connectors).

The temperature sensor is a bit more difficult to get working because it needs onewire support in the kernel. In my case the ATV kernel at that time did not have onewire support enabled:

```
$ zcat /proc/config.gz | grep -i W1
# CONFIG_W1 is not set
```

So check out the upcoming section about kernel compilation and flashing.

Once your programs are running correctly, you need to start them up when Linux starts (Linux Deploy takes care of starting it when Android starts if you enable autostart in Linux Deploy's settings). Unfortunately, since Linux Deploy runs in a chroot if you use the service command under Linux to start and stop services you will get a notice that It is running under a chroot and will not do anything. In my case, I needed only two scripts, so I had to create sys-v scripts for my processes. You can get a script template from https://github.com/fhd/init-script-template. Create /etc/init.d/pir-mqtt-agent, configure it to point to your script, make it executable and link it to /etc/rc3.d:

```
# chmod a+x /etc/init.d/pir-mqtt-agent
# cd /etc/rc3.d/
# ln -s ../init.d/pir-mqtt-agent S10pir-mqtt-agent
```

SYS-V may seem archaic, but it was fine for the past 20 years, and it is easier to understand than systemd. However, what if the scripts fail for some reason (e.g. the sensor is no longer there, network is offline, etc)? Without systemd to do service housekeeping there is nobody there to restart your dead scripts. I went the quick and dirty route and set up cron to periodically restart the scripts (even if they are not dead). That works fine for my needs, but this needs to be looked into, to create a more robust process management system.

```
# crontab -l
0 * * * * /usr/sbin/service pir-mqtt-service
restart
```

Compiling the Android kernel

Compiling the Android kernel is almost the same as compiling the Linux kernel, but it has some quirks. Some of the most important changes are:

- compilation is done on a x86_64 system (cross compilation)
- the zImage needs to be combined with a initrd and packed into a 16M boot image
- the boot image needs to be flashed to the Android boot partition
- the dtb lives in a dtbs partition and needs to be flashed too in case you are making changes to it.

First you will need to set up your cross compilation environment. Assuming you are on a Ubuntu 18.04 x86_64 system you can:

```
$ sudo apt-get -y install bc curl gcc git libssl-
dev
  libncurses5-dev lzop make u-boot-tools device-
tree-compiler
$ mkdir ~/toolchain
$ cd ~/toolchain
$ wget
https://releases.linaro.org/components/toolchain/b
inaries/6.3-2017.02/aarch64-linux-gnu/gcc-linaro-
6.3.1-2017.02-x86_64_aarch64-linux-gnu.tar.xz
$ unxz gcc-linaro-6.3.1-2017.02-x86_64_aarch64-
linux-gnu.tar.xz
$ tar xvf gcc-linaro-6.3.1-2017.02-x86_64_aarch64-
linux-gnu.tar
```

Next, download the Android kernel source:

```
$ mkdir ~/development
$ cd ~/development
$ git clone --depth 1
https://github.com/hardkernel/linux.git -b
odroidn2-4.9.y-android
$ cd linux
```

Now, set some environment variables so that the build environment knows that it needs to use the cross compiler. You will need to rerun these commands each time you start a new terminal (they are not persistent):

```
$ export ARCH=arm64
$ export CROSS_COMPILE=aarch64-linux-gnu-
$ export PATH=~/toolchain/gcc-linaro-6.3.1-
2017.02-x86_64_aarch64-linux-gnu/bin/:$PATH
```

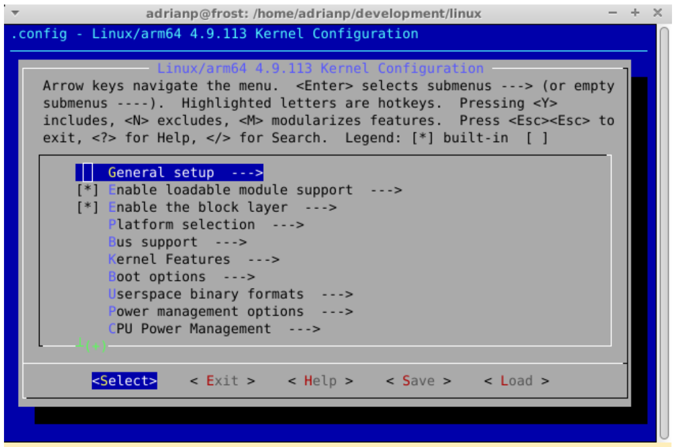
You can now start from a base configuration - usually by doing this inside the Linux directory:

```
$ make odroidn2_android_defconfig
```

If you want to start with a different config, you can extract the current kernel configuration from the running Android system from /proc/config.gz, gunzip it and save it as ~/development/linux/.config.

You can now run

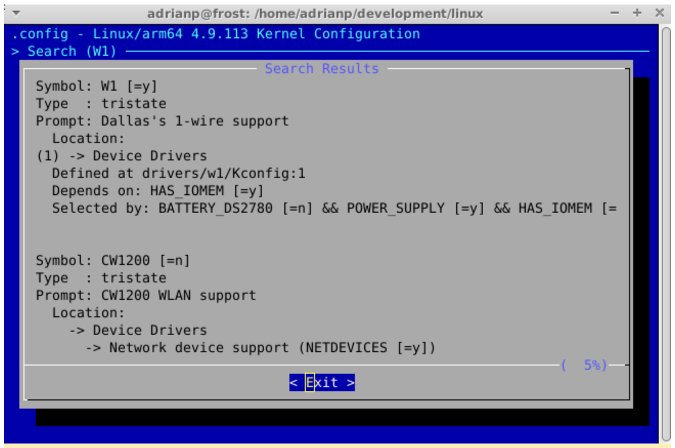
```
$ make menuconfig
```



```
adrianp@frost: /home/adrianp/development/linux
.config - Linux/arm64 4.9.113 Kernel Configuration
                 Linux/arm64 4.9.113 Kernel Configuration
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty
  submenus ----).  Highlighted letters are hotkeys.  Pressing <Y>
  includes, <N> excludes, <M> modularizes features.  Press <Esc><Esc> to
  exit, <?> for Help, </> for Search.  Legend: [*] built-in  [ ]

        General setup  --->
    [*] Enable loadable module support  --->
    [*] Enable the block layer  --->
        Platform selection  --->
        Bus support  --->
        Kernel Features  --->
        Boot options  --->
        Userspace binary formats  --->
        Power management options  --->
        CPU Power Management  --->
      ↓(+)

        <Select>    < Exit >    < Help >    < Save >    < Load >
```

Figure 3 - make menuconfig

Here you can search for the items that you need and enable them. For efficiency reasons you can find where a configuration option is in the menu by typing '/' and typing in the option name (e.g. in my case W1 - short for onewire). The output will tell you where you can find that option (in my case under Device Drivers), what its current state is and if it depends on other options (and what their states are). If you cannot find an option in the menu it is most likely hidden because one if its dependencies isn't met. By using the search function you can find which dependency is missing.



```
adrianp@frost: /home/adrianp/development/linux
.config - Linux/arm64 4.9.113 Kernel Configuration
> Search (W1)
                          Search Results
  Symbol: W1 [=y]
  Type  : tristate
  Prompt: Dallas's 1-wire support
    Location:
  (1) -> Device Drivers
    Defined at drivers/w1/Kconfig:1
    Depends on: HAS_IOMEM [=y]
    Selected by: BATTERY_DS2780 [=n] && POWER_SUPPLY [=y] && HAS_IOMEM [=


  Symbol: CW1200 [=n]
  Type  : tristate
  Prompt: CW1200 WLAN support
    Location:
      -> Device Drivers
        -> Network device support (NETDEVICES [=y])
                                                              (  5%)
                              < Exit >
```

Figure 4 - searching kernel options

Once you get to the menu you can find the item by pressing ALT and the highlighted shortcut letter and you should navigate through all options in the visible menu that share the same shortcut.

When enabling an option you generally have two choices - built-in or as a module. Traditionally, Linux uses modules and has the capability to auto-load modules on demand, but Android is a bit different and though it can use modules it does not auto-load them. The easiest choice is to build it in, but if you need too many options it might make the kernel grow in size too much and it will not fit (together with the initrd) in the 16M boot partition.

If you will build modules you will need to manually locate the module in the build environment (e.g. drivers/w1/masters/w1-gpio.ko) and copy it to /vendor/lib/modules/ on the Android side. Note that the /vendor partition is mounted read-only and needs to be remounted before copying:

:/ # mount -o remount,rw /vendor $ adb push drivers/w1/masters/w1-gpio.ko /vendor/lib/modules/

To load new modules at startup you will need to edit init.odroidn2.system.rc as described here: **https://github.com/codewalkerster/android_device_hardkernel_odroidn2/blob/s922_9.0.0_master/init.odroidn2.system.rc#L96**

For my needs (onewire support) I needed to enable the following modules (and have them built-in):

```
CONFIG_W1
CONFIG_W1_MASTER_GPIO
CONFIG_W1_SLAVE_THERM
```

Ok, now that the kernel configuration is done, you can save it and start building the kernel:

```
$ make -j$(nproc)
```

The compilation should take a while the first time, but subsequent runs should compile only the differences. Have a look for any errors, but if it builds correctly you should get a ~/development/linux/arch/arm64/boot/Image.gz.

```
$ ls -l
~/development/linux/arch/arm64/boot/Image.gz
```

If you need to make changes to the dtb (e.g. enable or change settings), you can use fdtput (part of device-tree-compiler package) to make changes to the dtb. In my case I need to explicitly enable one-wire support in the dtb, otherwise the driver will not do anything:

```
$ fdtput -t s
~/development/linux/arch/arm64/boot/dts/amlogic/me
son64_odroidn2_android.dtb /onewire status okay
```

Once the kernel and dtb are in order, we need to prepare the bootimg. To do this, first extract the boot partition from the working ATV version, so that we can keep the initrd from the original image. Next we install bootimg which is a tool for packing and unpacking Android boot images.

```
$ adb shell
# if=/dev/block/boot of=/dev/shm/boot.img
# exit
$ mkdir ~/development/bootimg
$ cd ~/development/bootimg
```

```
$ adb pull /dev/shm/boot.img
$ sudo apt-get install abootimg
```

We can now inspect the boot.img and extract the initrd and configuration:

```
$ file boot.img
boot.img: Android bootimg, kernel (0x1080000),
ramdisk (0x1000000), page size: 2048, cmdline
(otg_device=1 buildvariant=userdebug)
$ abootimg -x boot.img
```

The final step is to create the new bootimg with the new kernel:

```
$ abootimg --create boot-new.img -f bootimg.cfg -k
~/development/linux/arch/arm64/boot/Image.gz -r
initrd.img
```

Ok, now you are ready to flash the new kernel and dtb:

```
$ adb push boot-new.img /dev/shm
$ adb push
~/development/linux/arch/arm64/boot/dts/amlogic/me
son64_odroidn2_android.dtb /dev/shm
$ adb shell
# dd if=/dev/shm/boot-new.img of=/dev/block/boot
# dd if=/dev/shm/meson64_odroidn2_android.dtb
of=/dev/block/dtbs
```

Once you reboot your new kernel should take over and you can check that you get the functionality that you need. If it does not reboot you will need to recover the board by flashing the previous boot.img by using fastboot mode. Enjoy your Linux server with an Android TV GUI!

Shall We Play A Game? - Nintendo Drags Pokémon Into the 21st Century...Kicking and Screaming

April 1, 2020 By Dave Prochnow Gaming, Tutorial

Imagine being able to share, trade, and move your Pokémon between your games on different devices via a centrally-located, Internet-connected server. Sounds like some really wacky, far-out, futuristic stuff, eh? Well, maybe if this headline screamer had been announced in early 1999, it might've been. Rather, this is a completely valid "service" announcement that has been recently released by Nintendo. They even have the temerity to label this cloud service as Pokémon HOME.

Located at http://home.pokemon.com, the website offers an introduction to Pokémon HOME, as well as an explanation of the two available subscription services that details the differences between the Nintendo Switch and the Android versions. To use the Android version and take advantage of the new cloud service, you will need to go to Google Play and download the free Pokémon HOME app and subscribe.

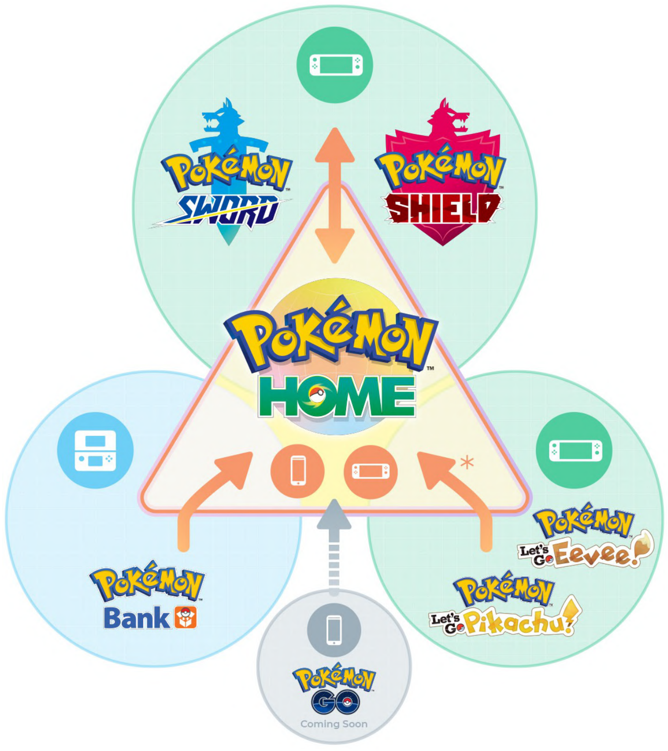


Figure 1 - Pokémon HOME

Right off the bat, you have a choice for your subscription plan: free or paid. The free plan is feature-rich enough to use right away with your Pokémon collection. You can easily opt for the free plan and test the Pokémon HOME waters before you jump in with your credit card.

The available plans are Basic (the free plan) and Premium (the paid plan). The Premium plan subscription is available for one month at $2.99, three months for $4.99, and one year for $15.99. In addition to differences between the Basic plan and the Premium plan, there are also platform feature differences between the Nintendo Switch and Android. Some of the feature differences between the plans and platforms:

1. Moving Pokémon from the 3DS-sponsored Pokémon Bank is only available with a Premium subscription. This is an included feature for Android.

2. The number of Pokémon that can be deposited into Pokémon HOME are limited to 30 in the Basic plan and 6,000 in the Premium plan.

3. The number of Pokémon that can be placed into the Wonder Box feature at one time are restricted to 3 for Basic users and 10 for Premium users.

4. The number of Pokémon that can be placed in the GTS (aka, Global Trade System) at once are 1 with Basic and 3 with Premium. This feature is also enabled for Android.

5. Basic users can only use pre-made rooms in Room Trade, whereas Premium uses can also make new user rooms. Android users will have the ability.

6. Only Premium subscribers can use the Judge function. This feature is available for Android.

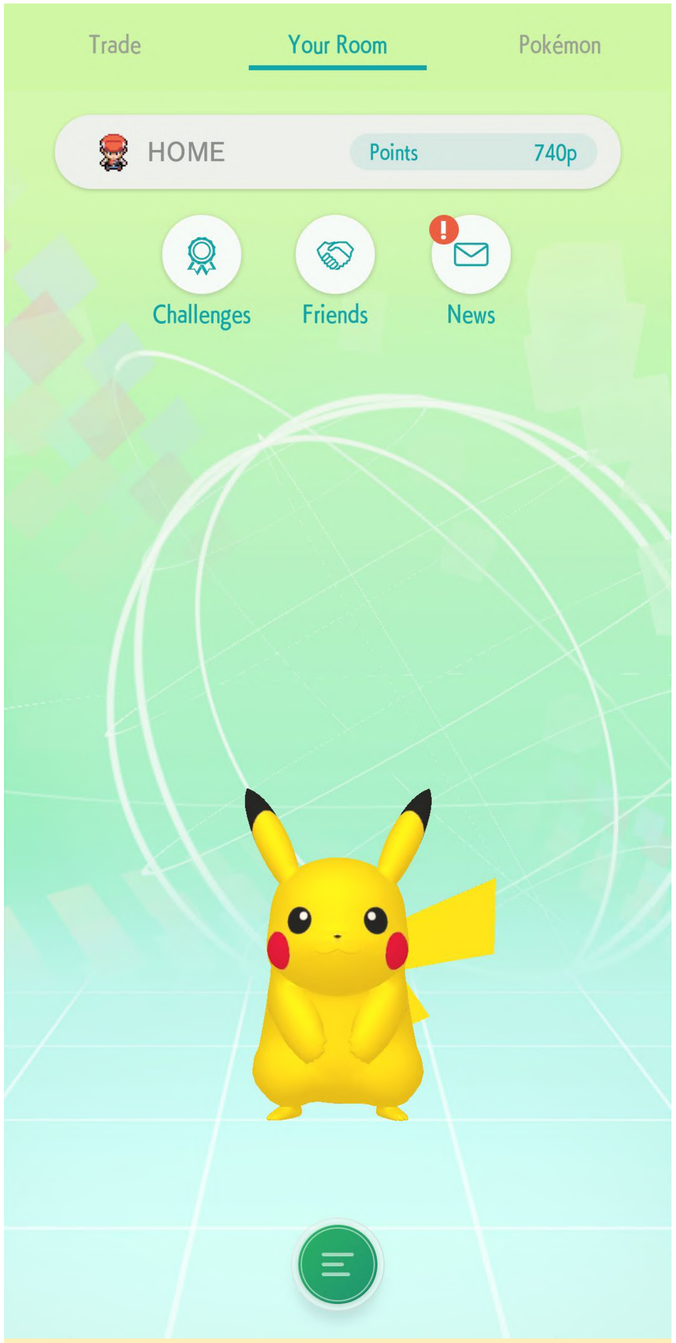


Figure 2 - Inside your Room

At present, the Nintendo Pokémon HOME Android version only supports Nintendo 3DS Pokémon Bank transfers, but NOT Pokémon Let's Go Eevee! or Pokémon Let's Go Pikachu! Although there is some speculation that Pokémon GO integration will soon be released for mobile devices. Therefore, smart ODROID game players might want to opt for the free Basic plan until the Pokémon GO option has been delivered and then decide whether to continue with the Basic plan or upgrade to the Premium plan.

Along with the Pokémon intra-game transfers of Pokémon HOME, there are also features for trading Pokémon via the Global Trade System and Room

Trade events, the ability to examine Pokémon in the Wonder Box, the opportunity to receive free "mystery" gifts, and convert your accumulated points into gaming Battle Points (BP). While Android users will be able to receive Mystery Gifts, Battle Point exchange is limited to the Nintendo Switch version only.

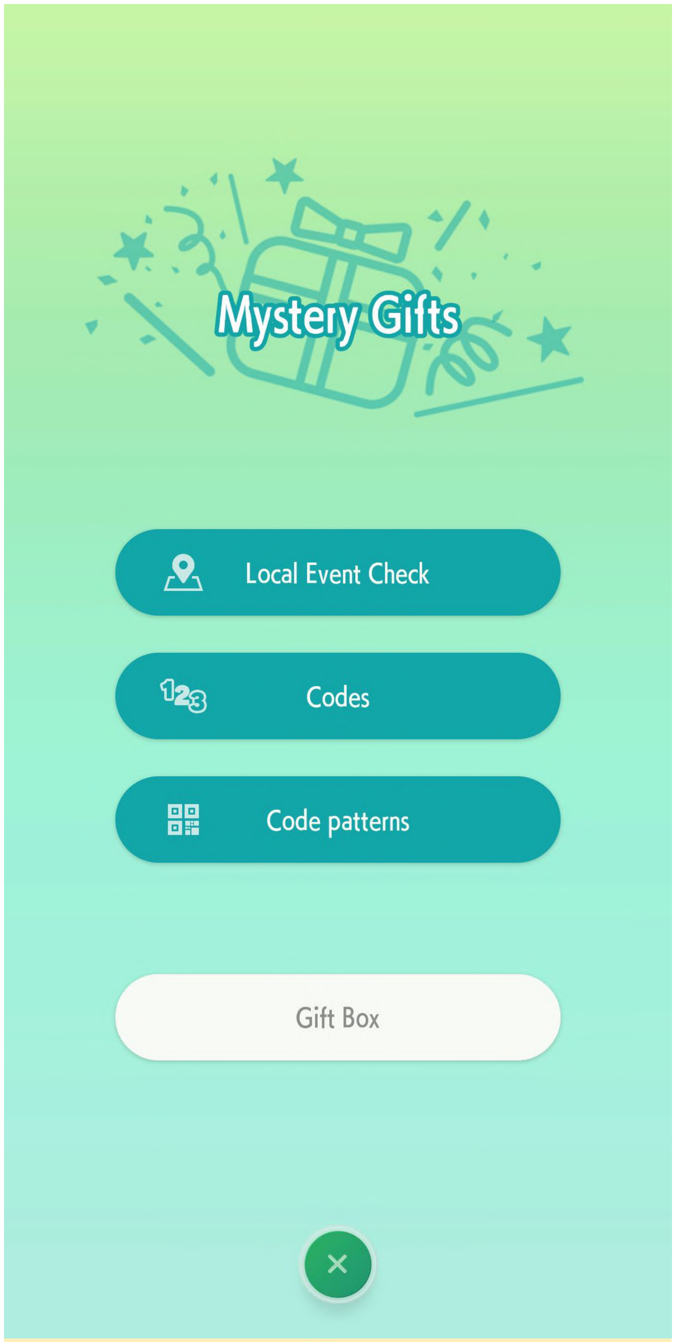
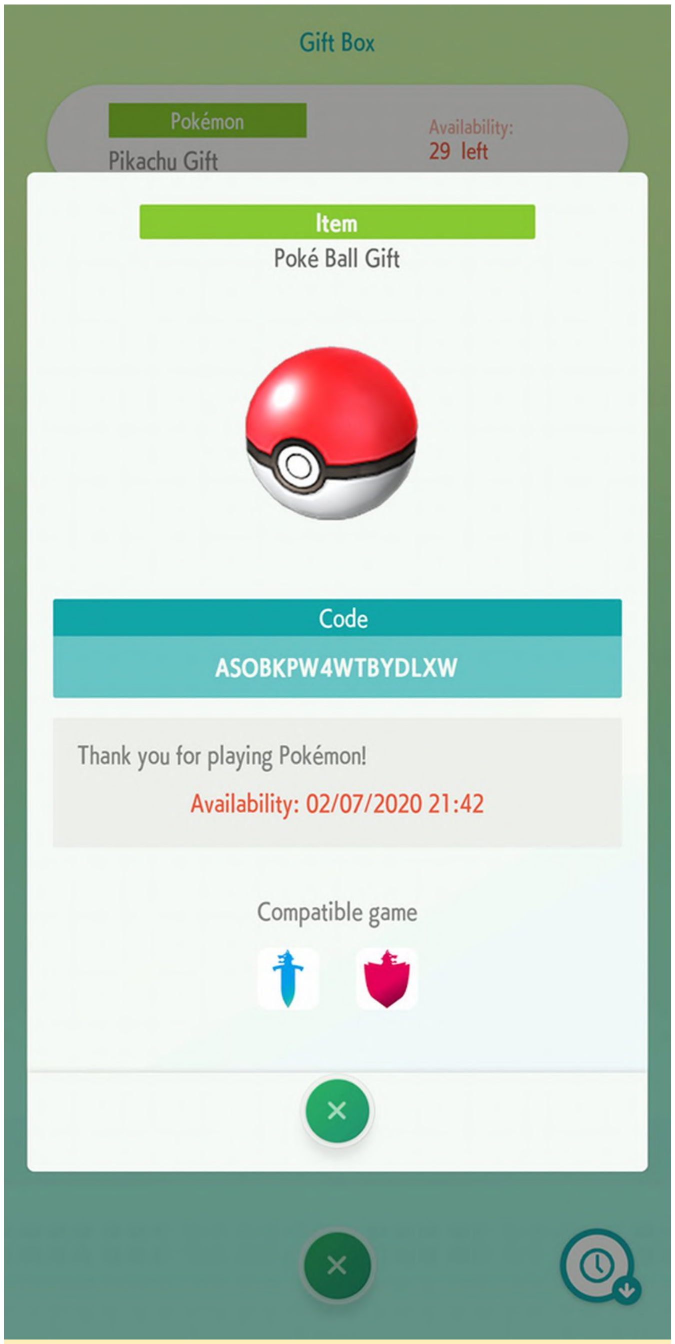


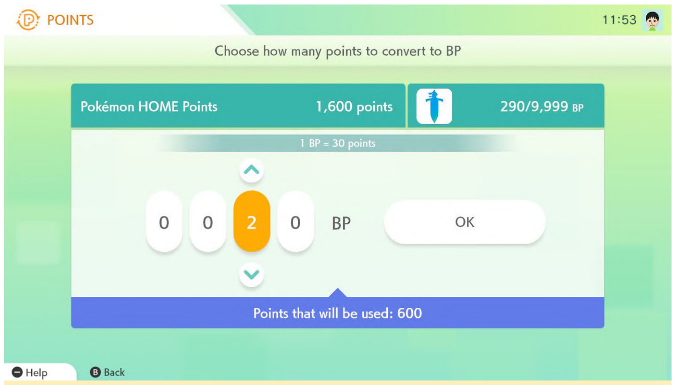
Figure 3 - Woo-hoo



Thank you for playing Pokémon!

Availability: 02/07/2020 21:42

Figure 4 - Mystery Gift



Choose how many points to convert to BP

Pokémon HOME Points 1,600 points | 290/9,999 BP

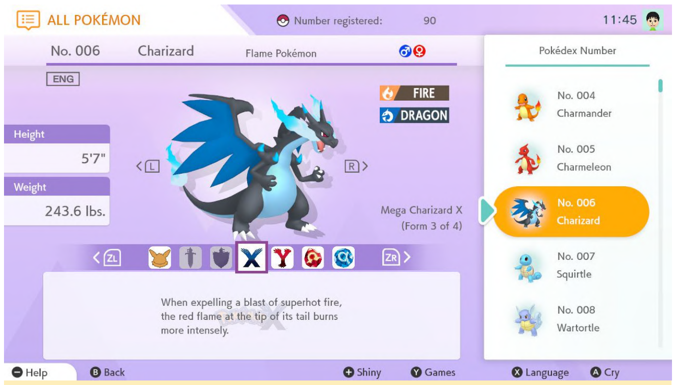
1 BP = 30 points

0 0 2 0 BP OK

Points that will be used: 600

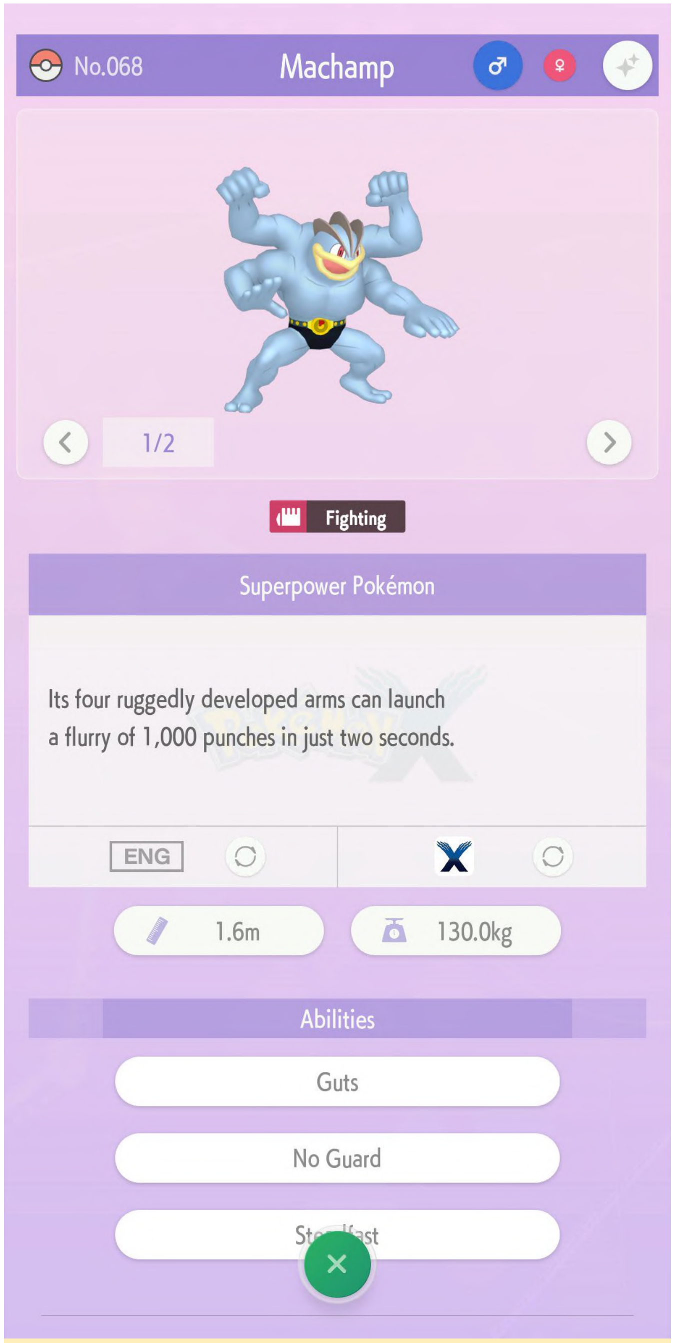
Figure 5 - Converting your points

One last feature worthy of mentioning is inside Pokémon HOME there is the ability to register all of your added Pokémon to the National Pokédex. Inside this dictionary, you can read about specific Pokémon abilities and moves that Pokémon can learn. This is a charming and informational inclusion that is found inside the Android version. You can also register both Mega Evolve and Gigantimax forms in this mobile database.



When expelling a blast of superhot fire, the red flame at the tip of its tail burns more intensely.

Figure 6 - National Pokédex



Its four ruggedly developed arms can launch a flurry of 1,000 punches in just two seconds.

Figure 7 - My Pokémon

While it seems like Nintendo took an eternity to catch-up to the rest of the gaming industry's embrace of cloud-based distributed game share, Pokémon HOME is a feature-rich product that has the ability to enable widespread Pokémon ecosystem adoption between active players. Even more remarkable, this shared experience could grow over 1000-fold with the arrival of the proposed Pokémon GO integration. That translates into seeing even more people wandering around aimlessly looking at their smart device searching for another Pokémon gym. Thanks Nintendo and welcome to the 21st century.

Building An Xbox Using an ODROID-H2

April 1, 2020 By @teeedubb ODROID-H2, Tutorial

Building an Xbox clone is something that I have wanted to do for a long time, but I never really got around to it for various reasons - and now it is finally happening. This is essentially a build log, because I know if I do not start it now, all you would get is a couple of photos of a finished build.

Figure 1 - I built an Xbox clone using an ODROID-H2

Why am I doing this? The OG Xbox rocked our world - we still call the media player 'the xbox' to this day. After it was modded it became a media powerhouse that was way ahead of its time. Loading games off the HDD, emulators and of course, XBMP. Unfortunately, due to the march of technology, it had to be retired. With my s912 box giving me headaches and the board I wanted was finally back in stock, I decided it was time to start this project and get the Xbox back into the lounge room.

The hardware I will be using is an ODROID-H2. I chose this because of the Intel GPU which has great support under Linux, with HDR support coming soon. Also, the x86 CPU outperforms many ARM systems and opens up more software possibilities, such as emulators, Firefox, Steam and Project M visualisations (it is not a media player if it does not have Project M!). The board has 2 video outs (HDMI2.0 and DP++) and with a passive DP to HDMI adaptor the DP++ port becomes a HDMI port which allows me to send 4k60 video to my TV and send HD passthrough audio to my HDMI 1.4 AVR at the same time. The board runs cool, has an external PSU and is small which will allow me to integrate a few things into the case and keep it uncluttered and quiet at the same time. I did think about a mini-itx board, but the ODROID-H2 has almost everything I need at a reasonable price.

The plan is to keep the outside fairly stock. I am not going to use an optical drive, so the DVD faceplate will be attached to the case. The LED will be changed to white and the front buttons will be used for power and reset. The smaller button has a power imprint and the larger one has an eject imprint and I am going to fill those up, paint them a similar colour and leave them blank; because I want to use the big button for power. I thought about attaching power and reset decals to the buttons, but I do not think that is really that important as 99% of the time a remote will be used to power it on. The left two controller ports will be covered with some smoked acrylic, with a IR receiver hiding behind one and I did think about putting a headphone port on the other, but I have never had the need for one and I have a USB soundcard or a USB DAC and amp that can be used if I ever do need it. The right two controller ports will be a USB 2 and 3 port, which again I do not often need, but they'll be handy for the occasional USB drive, wired controllers and charging the DS3 controllers I will be using with it. I did think about adding some kind of Kodi branding to the front, but I think the textured surface would make it difficult to get a good finish, so I will probably skip that.

Inside will be the board, 2x 4gb RAM sticks, a 60gb SSD for the OS and a 2.5" 320gb HDD for some backups and media storage. These are drives that I had as spare parts, and the 320gb will fill up very quickly so it will probably get upgraded pretty soon, maybe to an old 3.5" HDD I have. I will be using the NVME slot for an Intel WiFi plus BT 4 card (7265ngw) via an adapter, which will allow me to use dual large internal antennas and provide a good BT range for controllers. There will also be a 4 port USB3 hub inside to provide ports for the IR receiver, BT module, an arduino to drive TV backlighting and a Logitech unifying receiver. The front USB ports will be directly connected to the rear ports, which will leave 1 spare USB2 port on the rear. The ODROID-H2 is about 10x10cm and the Xbox is about 35x30cm so I do not see any issues fitting all that, just cable management. I purchased a 5V 92mm fan with the ODROID-H2 and it is too big to fit in the same location and the original fan, so I am going to mount it to the heatsink. It will overhang a bit so it will also move some air around the case which should help keep other components cool. I will keep an eye on temps and add a rear fan if need be, but many people run these fanless without issue so I do not think it will be a problem.

The operating system used will be Xubuntu 19.10, for now, and I will upgrade it to the 20.04 LTS version and keep it on LTS versions, afterwards. Since multiple applications will be used, I wanted a simple, light and customisable desktop environment. I generally run LTS versions for these types of applications, but I had some stuttering issues at first and tried a few versions of Xubuntu + Ubuntu + Libreelec while ironing out the issue and finally got it fixed on 19.10 and with 20.04 so close I decided to stay with it because I did not want to do another reinstall. If I knew what I know now I would stick with 18.04 because it has more precompiled software available (e.g., attract mode, emulationstation and an emulation PPA I found).

Since it is a dual monitor setup, though not in the traditional sense as the two monitors are the one TV, I have disabled use of the AVR monitor through some xrandr trickery because having a second monitor that was not easily accessible was really annoying. Mirroring does not work because of the different max resolutions of both monitors and disabling the second monitor would also disable sound. The only issue I have with this now is that the alt-tab switcher shows up twice, which is not a big issue because it will really only run one graphical program at a time, but a quick google shows that there are some possible alternatives for workarounds.

So far I am really impressed with this little machine and I highly recommend it for HTPC use - it plays 4k60 content smoothly, HDR coming soon and I do not have to buy a new AVR to get 4k video with passthrough audio. I have a script which automatically switches audio outputs depending on if the AVR is on or off and unlike the s912 box, it does not send audio to multiple outputs at the same time which makes operation simpler. 3D performance is lacking, but I was always expecting that. At 4k the Kodi GUI + Project M lag, but they're smooth at 1080p. At 1080p, the 3D map of Broforce is choppy, but the 2D sections are smooth. Super Meat Boy runs smoothly at 1080p. I tried Portal 2 but it loads to a blank screen. I have read that using DXVK can improve performance, but I have not delved into that yet. Also, there is always Steam in home streaming for anything too demanding.

Finally, some photos of the ODROID-H2 in its temporary home. Next will be dismantling the Xbox and I will be finding a home for the unused parts.

Since I have been using the ODROID-H2 for a few months now here are my thoughts on the device:

As I mentioned above, I am using it as a HTPC and for my purpose the ODROID-H2 is amazing and ticks all the “right” boxes - flawless kodi 4k60 HDR playback (I have tested a libreelec test build and HDR works on this board), x86 CPU, emulation and steam, Web browsers, SATA and twin HDMI/DP++ ports, mainline Linux availability and probably more. The Intel J4105 CPU struggles somewhat with 3D graphics and heavier emulation (e.g., gzdoom would not run with 60fps at 1080p). I have found that 720p with low 3D-gaming is possible with simple games (e.g., LEGO Starwars runs ok but Serious Sam 3 BFE was unplayable). PSX, DC emulation runs full speed, but Super Mario Sunshine was really choppy and Mario Kart Wii ran close to 60fps, but new Super Mario was choppy.

Desktop usage with the SATA SSD is snappy enough, though, I have not really done anything too demanding on there. Steam in home streaming works well and uses vaapi, but Web browsers do not use vaapi so it will struggle to play 4k YouTube via a browser (there is a Chromium build with vaapi support, but it did not work with a local streaming site).

Temperature hovers around 50C, in my case, and the fan is silent (I have adjusted the fan curve by 10'). I only wish this board had a dual NVMe slot, like I’ve seen on some motherboards, so that it would accept both a NVMe and WiFi/BT card. This would give the ODROID-H2 great connectivity without the use of adapters. While this machine will last me several years, I hope that Hardkernel will continue making these types of boards and possibly Mini-STX motherboards with no CPU attached.

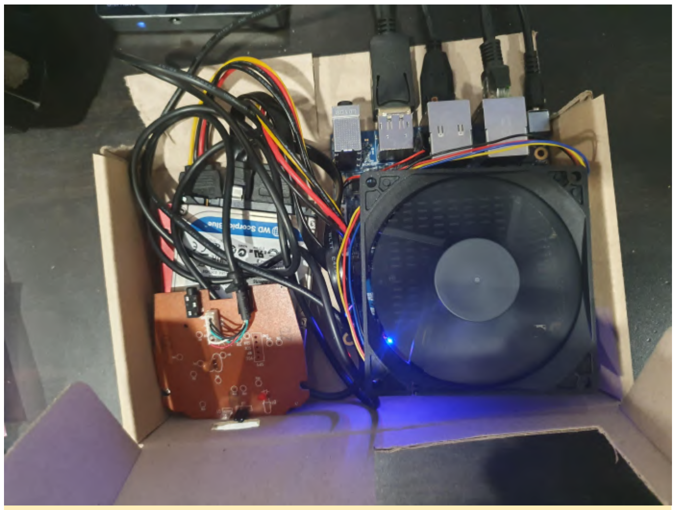


Figure 2 - Parts of the donor XBox which will be used

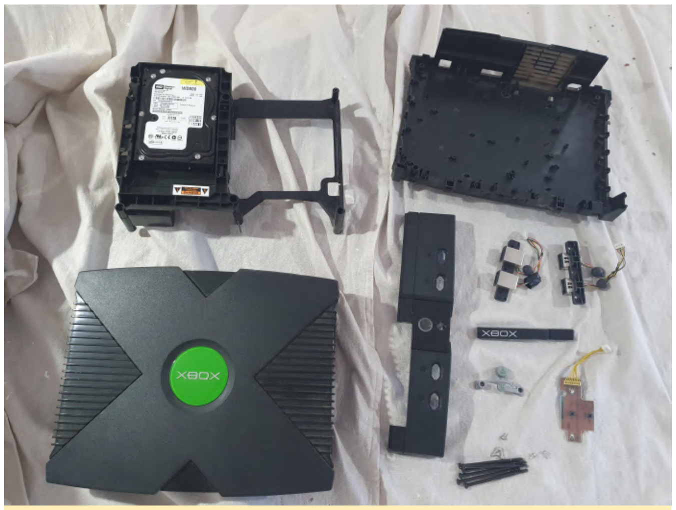


Figure 3 - Unfortunately, it powered up but did not output video, and the DVD-ROM would not open either

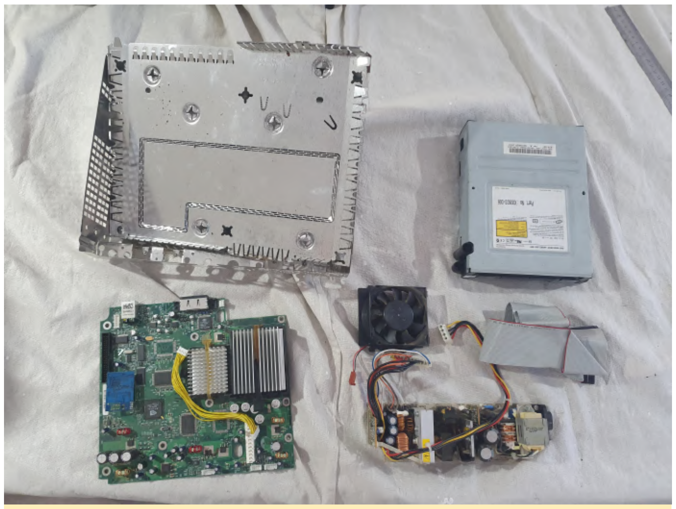


Figure 4 - Parts that I won't be using, with a trusty modchip installed. The console did not boot up, so I did not know what state these parts were in

Figure 5 - I bought it a week or two after release here. This is the best Console ever, in my opinion

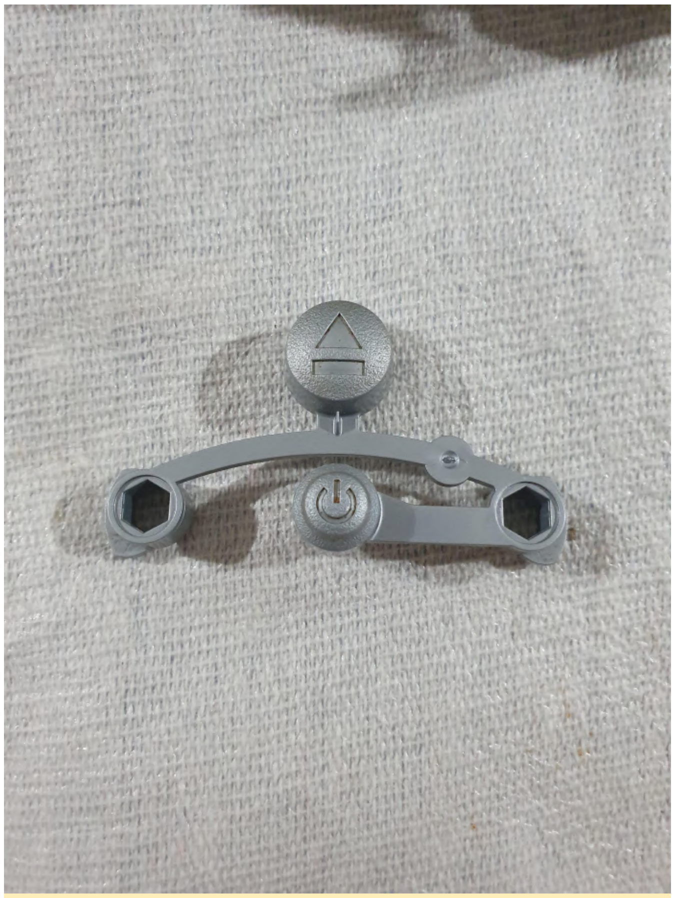


Figure 6 - First, I filed the front button and painted it. The buttons are grubby, but my workspace was clean at this point

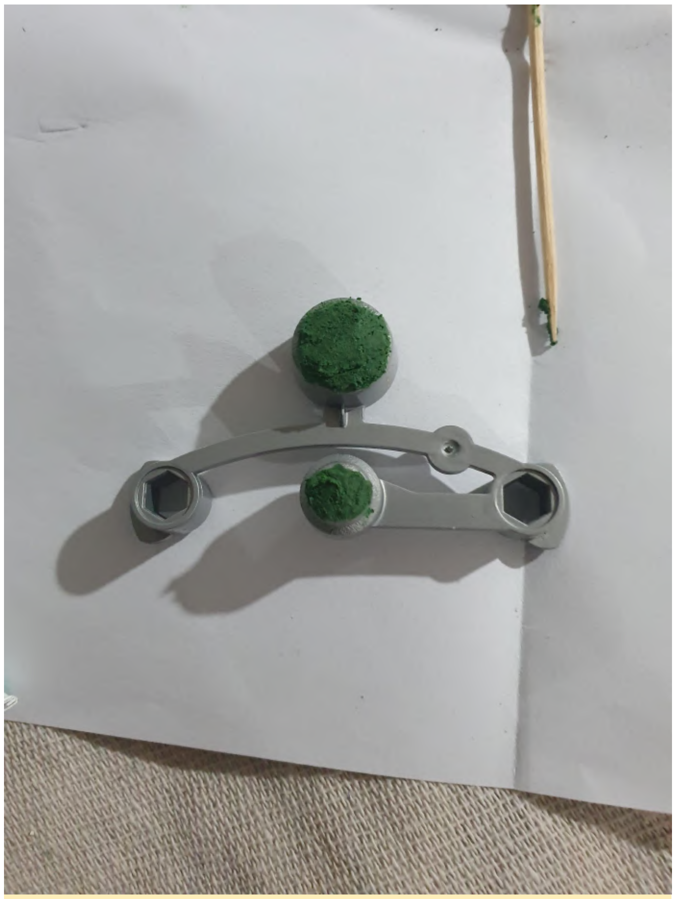


Figure 7 - Putty has been applied

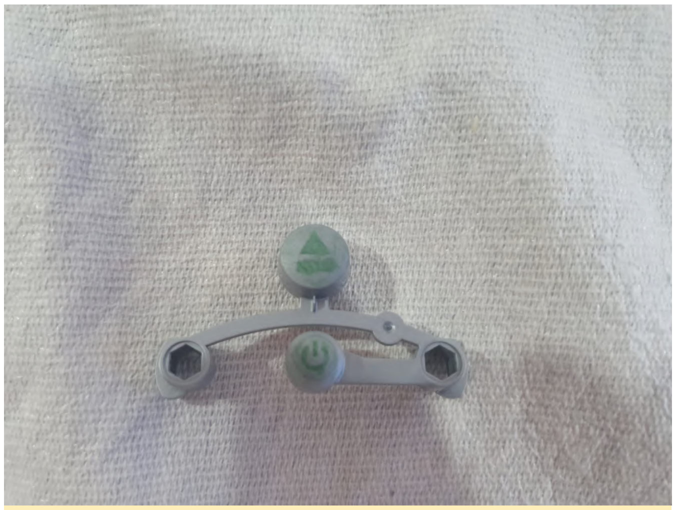


Figure 8 - This is how it looks, after two coats of putty and sanding down with a coarse emery board to keep the textured feel

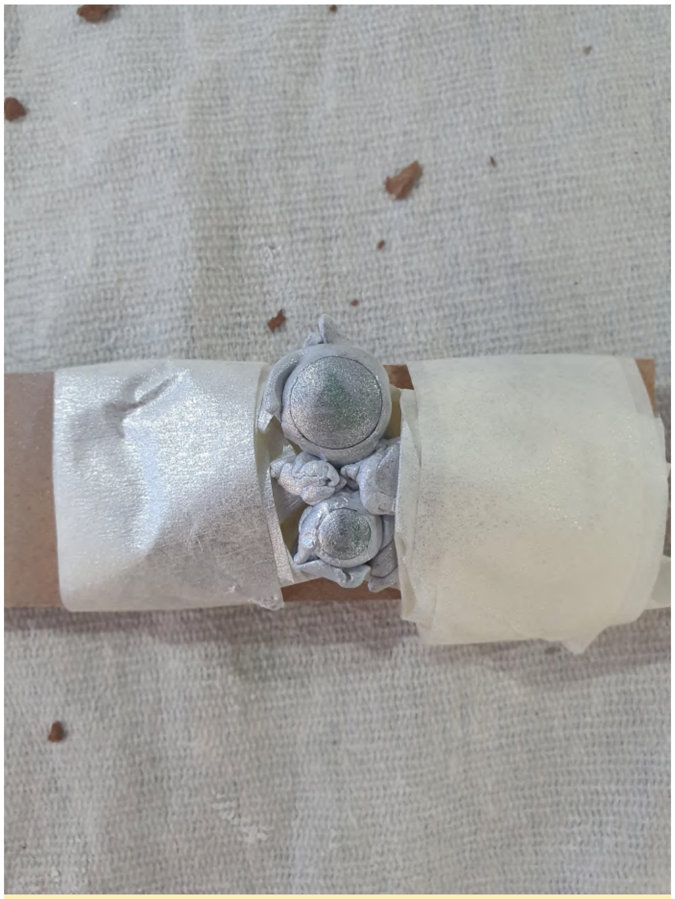


Figure 9 - This is how it appears after coating the buttons with 'satin silver'. It was a good match to the lid color

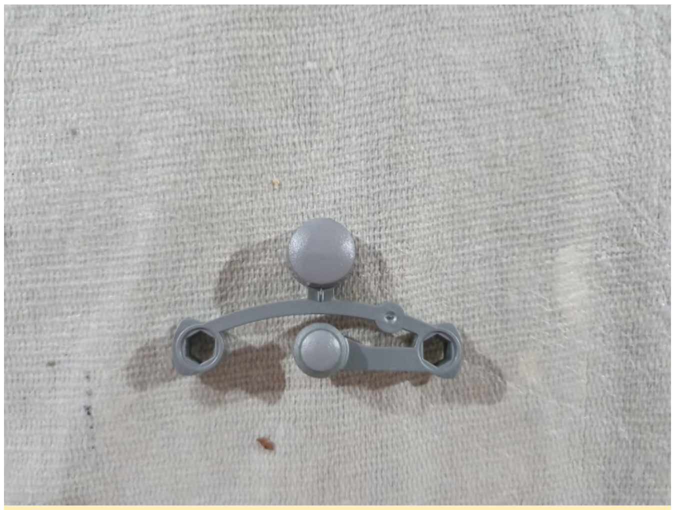


Figure 10 - I decided on a "mid grey" color instead of silver

The silver color turned out to be too bright so I switched to a 'mid grey'. The button now has a different texture and the colour is not a perfect match, but is close enough. Everything that can be seen when it is installed in the case, has been painted. You can see the outlines of the original buttons if you look close enough, but after about 50cm I cannot see them anymore.

References

https://imgur.com/gallery/oSOHuBt#LdS14Ui

Retro-Go: Yet Another ODROID-GO Emulator Launcher

April 1, 2020 By @ducalex Gaming, ODROID-GO

Retro-Go is a launcher with NES/GB/GBC/SMS/GameGear/Colecovision emulators. The emulator's code is based on Go-Play "Triforce" and the launcher's design is strongly inspired by pelle7's; however, it isn't a fork.

Features

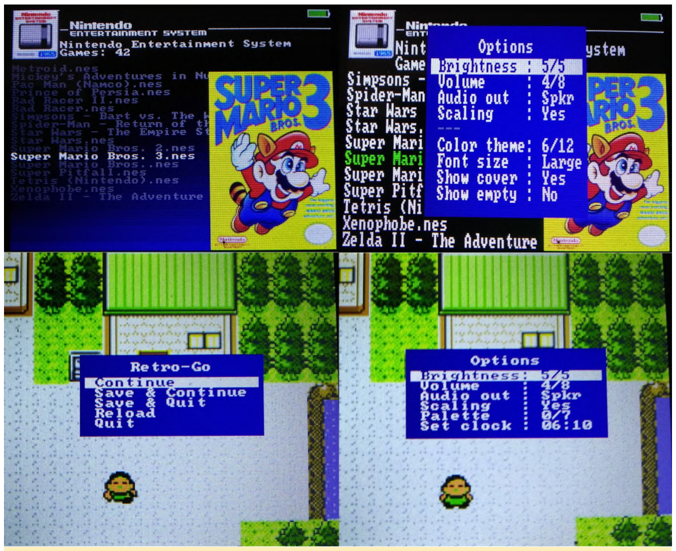
- In game menus
- Smoother/faster transitions
- Adjust RTC and preserve time between plays
- Launcher style is customizable
- Hide unwanted emulators
- NES PAL support
- Improved game compatibility
- More scaling options
- Bilinear filtering
- Fixed many crashes
- Easy to build
- And more!

Details

I encourage you to read the README (**https://github.com/ducalex/retro-go/blob/master/README.md**) to see the key bindings and more details.

Code, assets, and releases:

Repository: **https://github.com/ducalex/retro-go**
Releases: **https://github.com/ducalex/retro-go/releases**



```
Brightness : 5/5
Volume     : 4/8
Audio out  : Spkr
Scaling    : Yes
---
Color theme: 6/12
Font size  : Large
Show cover : Yes
Show empty : No
```

```
Brightness : 5/5
Volume     : 4/8
Audio out  : Spkr
Scaling    : Yes
Palette    : 0/7
Set clock  : 06:10
```

Figure 1 - Preview screenshots

For the original article and latest download link, please see the following post at: https://forum.odroid.com/viewtopic.php?f=159&t=37599

Retro Roller: An Optimized Gaming Experience For Your ODROID-GO Advance

April 1, 2020 By @valadaa48 Gaming, ODROID-GO Advance, Tutorial

Retroroller is a pre-built image for the ODROID-GO Advance that provides RetroArch 32bit and 64bit on CrashOverride's one and only stock image. Among other tweaks, a custom kernel is integrated that supports sleep. The idea with this image is to be able to update via a rolling package release instead of reflashing. Note that any major OS upgrades like CrashOverride's upcoming Ubuntu image will require a reflash. This image represents what I run on my ODROID-GO Advance and, as such, I will most likely not be making this into a kitchen sink of apps like other images.

Figure 1 - The Retro Roller Gaming Image for the ODROID-GO Advance is based on RetroArch

How to update

If you get prompted to update config files, choose which action you want to take. You should probably backup any files you changed first including RetroArch configuration files and scripts until you're comfortable with the dpkg upgrades. You only need to download the latest deb package and install that. This means you don't have to install each version up to the latest.

In the following commands, substitute for the most recent package version:

```
$ sudo dpkg -i retroroller_.deb
$ sudo apt-get update && sudo apt-get -f install
```

Key Bindings

There is a global listener with the following keybindings:

```
F3 + left/right -> volume
F3 + up/down -> brightness
power -> suspend
F3 + power -> shutdown
F3 + L -> perfnorm
F3 + R -> perfmax
```

RetroArch has specific keybinds as well:

```
F3 -> Hotkey
L + R -> Menu
```

You can use the “Customize Settings” -> “Input” option to change the keybindings to your liking.

Features

- 64bit and 32bit app support (thanks to CrashOverride's stock image)
- RetroArch 64bit and 32bit pre-installed
- Blue heartbeat led is turned off
- perfmax also sets the mem governor to high
- Custom kernel with sleep enabled
- RetroArch apps setup to download cores from Safarikniv's site
- NOirBRight's es_system.cfg uses retroarch where available
- Shutdown/reboot works
- Headphone/speaker auto-switching works

Credits

- CrashOverride for all of his hard work on creating the Stock image, libgo2,
- initial retroarch port and countless other things for this platform.
- Lakka team for their RetroArch patches, testing and integration.
- NOirBRight for his extensive testing and feedback.
- Safarikniv for hosting both 32bit and 64bit RetroArch cores for download.

Pull requests from the community are welcome!

Releases

The initial release and based image are available at https://mega.nz/#!etlnUTjT!nxOhiLzG03jbXgcyiUeh7_B6ovHBqvDFGWTQ4uR58Ho. Updates are available at https://github.com/valadaa48/retroroller/releases.

To update the image, ou need to have SSH access (via WiFi or ethernet). You have first to copy the .deb file onto the ODROID-GO Advance using WinSCP or Filezilla:

```
$ scp retroroller_1.0-2.deb odroid@OGA_IP_ADDRESS
```

Next, SSH to the ODROID using the Linux command line on another computer, or use PuTTY on Windows:

```
$ ssh odroid@OGA_IP_ADDRESS
```

Then, update the package:

```
$ sudo dpkg -i retroroller_1.0-2.deb
```

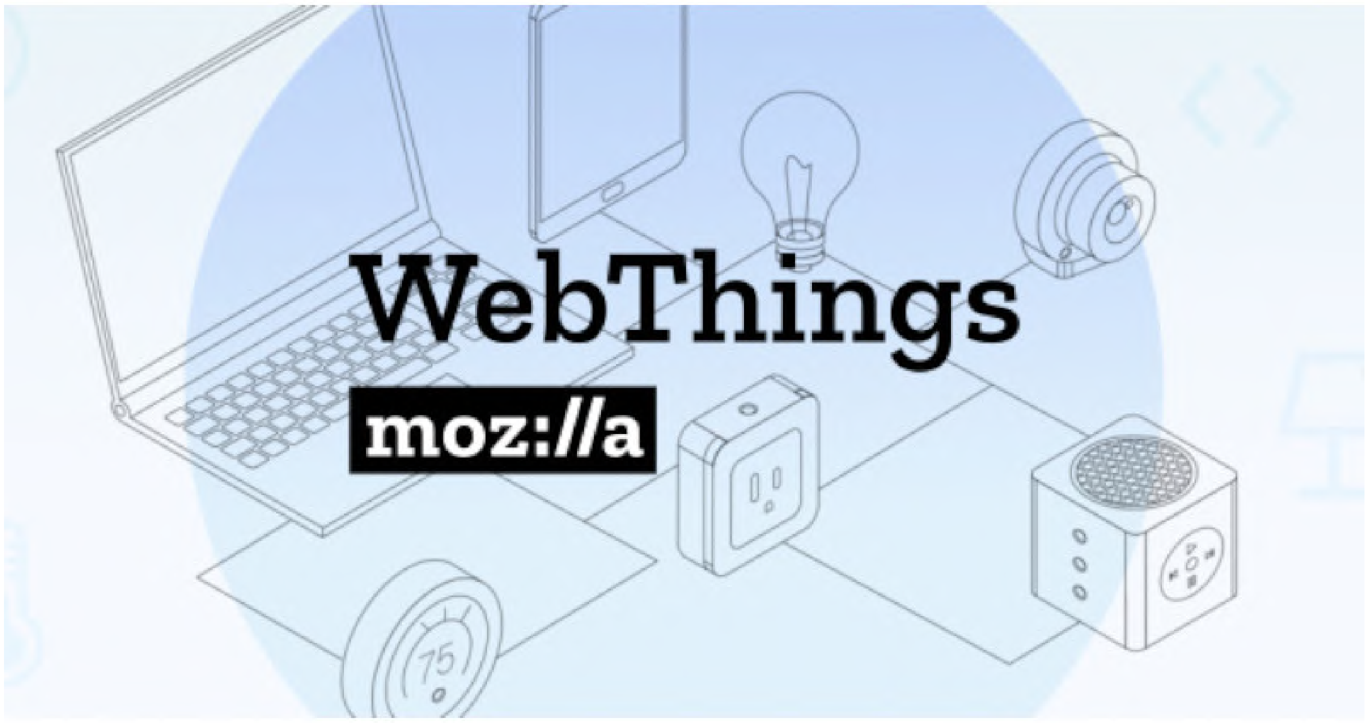
When asked if you want to overwrite retroroller_boot.sh, answer Y or I:

Configuration file '/opt/retroroller/scripts/retroroller_boot.sh' ==> File on system created by you or by a script. ==> File also in package provided by package maintainer. What would you like to do about it ? Your options are: Y or I : install the package maintainer's version N or O : keep your currently-installed version D : show the differences between the versions Z : start a shell to examine the situation The default action is to keep your current version. *** retroroller_boot.sh (Y/I/N/O/D/Z) [default=N] ? I Installing new version of config file /opt/retroroller/scripts/retroroller_boot.sh ...

The unofficial release thread is available at https://forum.odroid.com/viewtopic.php?f=193&t=38016. For comments, questions, and suggestions, please visit the original article at https://forum.odroid.com/viewtopic.php?f=193&t=38140.

WebThings on Armbian: Using the ODROID-XU4 for the Internet of Things

April 1, 2020 By Philippe Coval Development, ODROID-XU4, Tinkering



Mozilla WebThings is a platform for controlling home devices. A primary purpose is to give users the ability to set up and control a "smart home" on their own without relying on third party services. This article will tell you why and how to set it up on ODROID-XU4.

Why?

For years, cloud -operated devices have had limitations. Problems with interoperability and reliability are beyond the scope of this article, but one that needs to be addressed is privacy. I personally consider that taking or revealing (any) user's data without their explicit permission is a problem that everyone should consider. Even if you don't see yourself as having anything to hide, it is doubtful you'd want to share your credit card numbers, so it's important to be thoughtful and considerate before adopting a service or connecting devices because your privacy policy will also affect other family members or even your guests. We have seen some progress when it comes to strengthening privacy in the last couple of years thanks to certain regulations. Europe started with GDPR, which outlined some important regulatory guidelines when it comes to software that handles personal data. One important one specifically, article 25, requires "Privacy by Design" in technical systems. This means they must be designed to maintain users' privacy from the ground up.

Figure 1 - GPDR

Who?

It is reasonable to predict that the connected devices market will change sooner or later. For now, you can stay ahead and try FLOSS projects which did not wait for regulations to enable privacy. Since the dawn of the Mozilla IoT platform development, the project was explicitly designed to avoid your data going to someone else's cloud.

Figure 2 - NoCloud

The key principle is pretty simple: all home devices are connected only to the home network, so data just stays local, nothing is going out. Without going into too many details, there are many IoT standards today, and probably none will be appropriate to match every use case. That said, a common technology can abstract many overlaps. This is the mission of W3C, the organization known for the standardization of the world wide web (HTML, HTTP, and XML). W3C's "Web of Things" working group (WoT) is committed to address some IoT problems using existing web technologies and fill the missing gap. While some describe WoT as the "HTML for IoT", think that WoT protocols are mostly for machines instead of humans, technically we are talking about REST APIs and JSON Schema for semantics.

What?

The WebThings framework is based on simplified WoT proposed recommendations. Today, the implementation is composed of:

WebThings API for devices that speak REST on HTTP. WebThings gateway can control devices from a nice web application. WebThings gateway's add-on adapter(s) to translate non-WebThings devices or services to WebThings API.

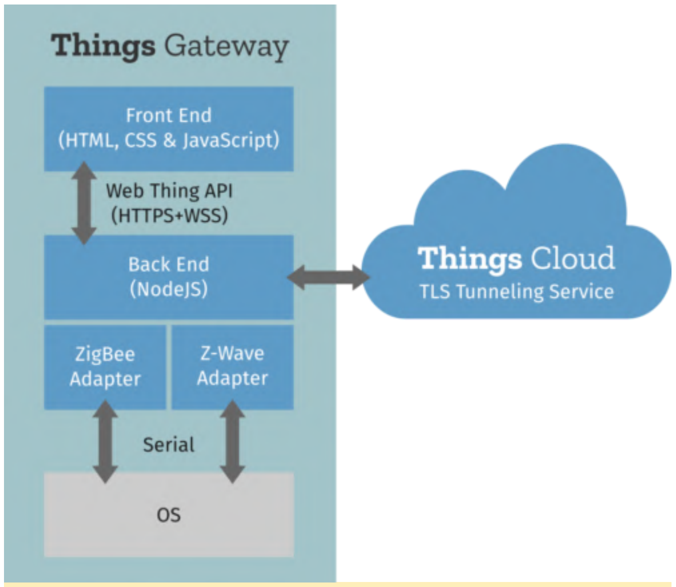


Figure 3 - ThingsGateway

How?

In this article, I will cover how an ODROID-XU4 can be a valid single-board computer for hosting WebThings gateway software which was originally developed for reference target Raspberry Pi 3.

Figure 4 - ODROID-XU4

Following these instructions should also work for other ODROIDs with minor adaptations. If you need to pick one, I suggest you consider the ODROID-H2, because it should be a bit simpler to develop on x86_64 CPU architecture. Setup is not straightforward so you'll need a GNU/Linux host computer, SD-card, Serial Port and Ethernet link.

Setup Armbian

The Mozilla reference image is based on Raspbian, a Debian port for Raspberry Pi, this image won't work on non-Pi devices. Hopefully the Armbian project has just released images to boot the latest Debian-10 on various ARM boards. Either the Debian or Ubuntu flavor of Armbian will work on ODROID-XU4, but let's download the minimal image and then dump it to an SD card.

On a GNU/Linux system, I used an USB SD adapter and these command lines:

```
$ lsblk # Will list your disks make sure to use
the right one
$ disk='/dev/disk/by-id/usb-Generic-
_USB3.0_CRW_-0_000000021716-0:0' #
$ file "$disk" # TODO

$
release="Armbian_20.02.1_ODROIDxu4_buster_current_
5.4.19_minimal"
$
url="https://dl.armbian.com/odroidxu4/archive/${re
lease}.7z"

$ sudo sync
$ sudo apt-get install curl 7zip-full time #
Install those tools on Debian or adapt
$ time curl -O "$url" # 3sec 155313070c
$ time 7z e -so "$release.7z" "$release.img" |
sudo dd bs=4MB of="$disk"
#| 562036736 bytes (562 MB, 536 MiB) copied,
142.735 s, 3.9 MB/s
```

Booting

Insert the SD image on ODROID-XU4. If you have the serial port configured you should be able to view the boot log:

```
$ screen /dev/ttyUSB0 115200

#| U-Boot 2017.05-armbian (Feb 17 2020 - 07:52:44
+0100) for ODROID-XU4
#|
#| CPU: Exynos5422 @ 800 MHz
#| Model: ODROID XU4 based on EXYNOS5422
#| Board: ODROID XU4 based on EXYNOS5422
#| Type: xu4
#| DRAM: 2 GiB
#| MMC: EXYNOS DWMMC: 0, EXYNOS DWMMC: 1
#| MMC Device 0 ( SD ): 7.4 GiB
#| mmc_init: -5, time 4
#| *** Warning - bad CRC, using default
environment
#|
#| In: serial
#| Out: serial
#| Err: serial
#| Net: No ethernet found.
#| Press quickly 'Enter' twice to stop autoboot: 0
#| (...)
```

A couple of minutes later, you should be able to log in as root with the default password "1234". Then system will ask setup default user:

```
#| Armbian 20.02.1 Buster ttySAC2
#| odroidxu4 login: root
#| Password: 1234
#| You are required to change your password
immediately (root enforced)
#| Changing password for root.
#| (current) UNIX password:
#| Enter new UNIX password:
#| Retype new UNIX password:
#| ___ _ _ _ __ ___ _ _ _
#| / _  __| |_ __ ___ (_) __| |  / / | | | || |
#| | | | |/ _` | '__/ _ \| |/ _` |  /| | | | || |_
#| | |_| | (_| | | | (_) | | (_| | / \| |_| |__ _|
#| ___/ __,_|_| ___/|_|__,_| /_/_\___/ |_|
#|
#| Welcome to Armbian buster with Linux 5.4.19-
odroidxu4
#|
#| System load: 2.23 0.79 0.28 Up time: 1 min
#| Memory usage: 6 % of 1993MB IP: 192.168.1.232
#| CPU temp: 32°C
#| Usage of /: 29% of 7.1G
#|
#| New to Armbian? Check the documentation first:
https://docs.armbian.com
#|
#|
#| Thank you for choosing Armbian! Support:
www.armbian.com
#|
#| Creating a new user account. Press  to abort
#| Desktop environment will not be enabled if you
abort the new user creation
#|
#| Please provide a username (eg. your forename):
user
#| Trying to add user user
#| Adding user `user' ...
#| Adding new group `user' (1000) ...
#| Adding new user `user' (1000) with group `user'
...
#| Creating home directory `/home/user' ...
#| Copying files from `/etc/skel' ...
#| Enter new UNIX password:
#| Retype new UNIX password:
#| passwd: password updated successfully
#| Changing the user information for user
#| Enter the new value, or press ENTER for the
default
#| Full Name []: User
#| Room Number []:
#| Work Phone []:
#| Home Phone []:
#| Other []:
#| Is the information correct? [Y/n] y
#| Dear User, your account user has been created
and is sudo enabled.
#| Please use this account for your daily work
from now on.
```

Once done, let's inspect the system, and install a multi-cast DNS service that will help to connect remotely using SSH:

```
root@odroidxu4:~# cat /etc/os-release
#| PRETTY_NAME="Debian GNU/Linux 10 (buster)"
#| NAME="Debian GNU/Linux"
#| VERSION_ID="10"
#| VERSION="10 (buster)"
#| (...)

root@odroidxu4:~# df
#| Filesystem 1K-blocks Used Available Use%
Mounted on
#| udev 950928 0 950928 0% /dev
#| tmpfs 204128 6592 197536 4% /run
#| /dev/mmcblk1p1 7505192 498264 6915480 7% /
#| tmpfs 1020628 0 1020628 0% /dev/shm
#| tmpfs 5120 0 5120 0% /run/lock
#| tmpfs 1020628 0 1020628 0% /sys/fs/cgroup
#| tmpfs 1020628 0 1020628 0% /tmp
#| /dev/zram0 49584 632 45368 2% /var/log
#| tmpfs 204124 0 204124 0% /run/user/0

root@odroidxu4:~# sudo apt-get update
root@odroidxu4:~# sudo apt-get install avahi-
daemon
root@odroidxu4:~# reboot
```

Once rebooted, let's login using SSH with hostname.local address instead of IP that may be different for each of us:

```
$ ssh root@odroidxu4.local
```

Extra storage

Optionally you can skip or use this trick to preserve SD-card's life span. I just plugged a pair of USB sticks (of 4GB each) in the two USB3 ports of ODROID. This extra USB mass storage will be used for swap memory and docker. once mounted as follow:

```
$ sudo=$(which sudo)

$ dev="/dev/disk/by-id/usb-Generic_Mass-Storage-
0:0" # TODO
$ file "$dev"
$ fdisk -l $dev || lsblk # Update previous line

$ dev='/dev/sda' # TODO update
$ label="docker"
$ yes | ${sudo} mkfs.ext4 -L "$label" "$dev" #
TODO: verify $disk variable

$ dev=/dev/sdb # TODO: update if needed
$ label="swap"
$ fdisk -l $dev
$ yes | $sudo mkswap -L "$label" "$dev"
$ free
#| total used free shared buff/cache available
#| Mem: 2041260 107724 1759992 6592 173544 1865772
#| Swap: 1020628 0 1020628
$ sudo swapoff -a
$ free
#
#| total used free shared buff/cache available
#| Mem: 2041260 106224 1761472 6592 173564 1867272
#| Swap: 0 0 0

$ sudo swapon "/dev/disk/by-label/swap"
$ free
#| total used free shared buff/cache available
#| Mem: 2041260 107912 1759716 6592 173632 1865584
#| Swap: 3943420 0 3943420
```

Install docker to use the other USB disk:

```
$ sudo apt-get install docker.io time git lsb-
release file
#| Need to get 55.9 MB of archives.
#| After this operation, 255 MB of additional disk
space will be used.
#| Do you want to continue? [Y/n] Y
#| (...)

$ dev="/dev/disk/by-label/docker"
$ mnt="/var/lib/docker"
$ df -h "$mnt"
# /dev/mmcblk1p1 7.2G 1.2G 6.0G 17% /
$ sudo systemctl stop docker
$ sudo sync
$ sudo mkdir -p "$mnt"
$ sudo mount "$dev" "$mnt"
$ df -h "$mnt" # /dev/sda 3.7G 16M 3.5G 1%
/var/lib/docker
$ sudo systemctl restart docker
$ sudo docker version # 18.09.7
```

Build binaries

There are various ways to use Mozilla WebThings platform. The simplest one would be to use the deb package built for Raspbian but ARMv6 version won't be optimized for our ARMv7 CPU. So let's try to build it again on the device using docker to make sure the whole process is replicable. It was for latest release 0.11.0, so I published "webthings-gateway_0.11.0-1_armhf-debian-buster.deb" package at https://bintray.com/rzr/deb/webthings-gateway#files.

Feel free to install this one or rebuild on the device using the following steps:

```
$ sudo apt-get install docker.io time git lsb-
release file
$ sudo apt-get install time screen
$ url=https://github.com/mozilla-iot/gateway-
deb.git
$ git clone --depth 1 --recursive "${url}"
$ cd gateway-deb
$ sudo time bash ./build-docker.sh

$ sudo docker image ls
#| REPOSITORY TAG IMAGE ID CREATED SIZE
#|   c0edf15b50a7 About an hour ago 122MB
#| gatewaydeb_default latest 32e5bdf8321c 8 hours
ago 843MB
#|   51a23a2e7130 9 hours ago 1.04GB
#| Debian 10 3eee7456d779 3 weeks ago 92.8MB

$ du -hsc ./dist/*
#| 21M ./dist/WebThings-gateway_0.11.0-1_armhf-
debian-buster.deb

$ sudo chmod -Rv 700 ./dist/WebThings-
gateway_*.deb
$ sudo apt install -y ./dist/WebThings-
gateway_*.deb
#| Need to get 0 B/40.5 MB of archives.
#| After this operation, 180 MB of additional disk
space will be used.
#| Do you want to continue? [Y/n] Y
#| (...)

$ sudo reboot
```

Webapp

Now the most difficult part is done (no more command lines!) Let's wear the user hat and connect to the dashboard by pointing your browser at http://odroidxu4.local:8080/. The welcome page should appear as follows:

```
| Mozilla IoT
| Welcome
| Choose a secure web address for your gateway:
| [subdomain].mozilla-iot.org
| [Email]
| [Create]
| [Skip]
```

The gateway can be registered on mozilla.org for remote management, this is optional, so let's skip subdomain as we won't use the gateway from the Internet for our first experiment. The next page will ask to create (local) credentials:

```
| Welcome
| Create your first user account:
| user: [user]
| email: [user@localhost]
| password: [password]
| password: [password]
| [Next]
```

And now we're ready to add some WebThings, and you can start filling out your dashboard with Virtual Resources. First hit the "burger menu" icon on the top left, go to settings page, and then go to the "add-ons" page at **https://odroidxu4.local/settings/add-ons/** and enable a "Virtual Things" adapter:

```
| Mozilla IoT Virtual Things Adapter
| by Mozilla IoT
```

Once enabled It should be listed on the adapters page at https://odroidxu4.local/settings/adapters. You can then go back to the 1st "Things page" (it's the first entry in the menu). We can start adding "things" by pressing the bottom "+" button at https://odroidxu4.local/things, then press "Done" at the bottom:

```
| Virtual On/Off Color Light
| Color Light
| Save
```

From this point, you can decide to control a virtual lamp from the UI, and even establish some basic rules (second entry in menu) with more virtual resources.

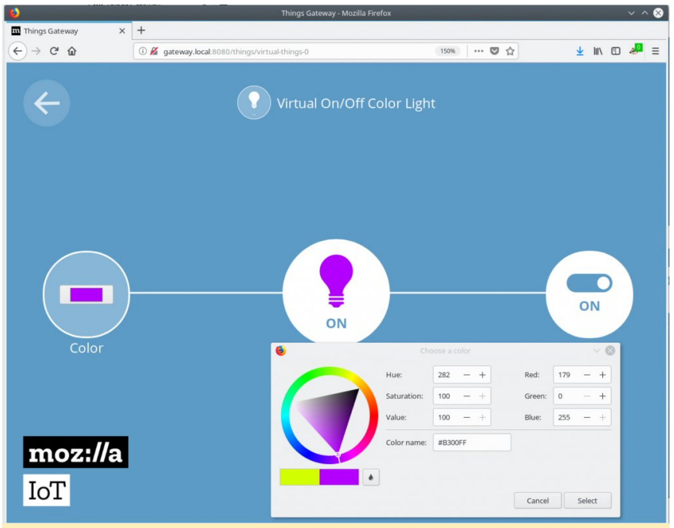

Things Gateway - Mozilla Firefox
Things Gateway
gateway.local:8080/things/virtual-things-0
150%
Virtual On/Off Color Light
Color
ON
ON
moz://a
IoT
Choose a color
Cancel
Select
Hue: 282
Saturation: 100
Value: 100
Red: 179
Green: 0
Blue: 255
Color name: #B300FF

Figure 5 - Virtual Things

Join the community

We just validated that our setup is working using mock devices, so the next step is to look at other add-ons. Today there are above 100, which is a great showcase of community contributions. A typical use of an add-on adapter is to connect a device that speaks another protocol, and the add-on is just translated to WebThings abstractions. The gateway can support Zigbee, ZWave devices you'll find on the market. That said, some add-ons might not work flawlessly on our ODROID-XU4 setup, so please report issues with related projects to @rzr on GitHub and I'll be glad to share hints or fixes.

It goes even beyond that. Any stuff that has an API can be managed using the WebThings platform. As an example, I made the Activity Pub adapter that allows posting a public message to mastodon social network when some conditions are met. Automation is possible using the rule engine where users can link sensors to actuators to establish "smart behaviors". Remote control is possible from a progressive web app served by the gateway itself. In my opinion, this is much simpler and trustworthy than when you are forced to install an app from a store on personal phones to get access to the devices you bought.

Mozilla's WebThings project is a good demonstration of how a service could look the same to users but is operated totally differently than what you'll find today on the IoT market. Your feedback is welcome, so that WebThings can be improved.

Resources

- https://iot.mozilla.org/
- https://blog.mozilla.org/blog/2018/02/06/announcing-project-things-open-framework-connecting-devices-web/
- https://fosdem.org/2020/schedule/speaker/philippe_coval/
- https://www.hardkernel.com/shop/odroid-xu4-special-price/
- https://www.armbian.com/
- https://edpb.europa.eu/our-work-tools/public-consultations-art-704/2019/guidelines-42019-article-25-data-protection-design_en https://mastodon.social/@rzr/103805535349436510
- https://twitter.com/RzrFreeFr/status/1237784387346989057

Modding Your ODROID-GO Advance for Wireless Charging: A Simple DIY Project

April 1, 2020 By @HiveTyrant ODROID-GO Advance, Tinkering, Tutorial

I added wireless charging capabilities to my ODROID-GO Advance using a wireless charging receiver and a USB breakout board. I happened to have the parts lying around but it literally only took a couple of minutes to do and was incredibly simple. The hardest part is taking the case off of the ODROID-GO Advance without breaking clips!

The wireless charging receiver can be found on all the usual sites, such as AliExpress, eBay, or Amazon, although Amazon is a bit more expensive. The USB breakout board was a couple of dollars from Adafruit, but can also be purchased from the other sites.

I'll probably add a USB hub or USB C charging next, but honestly, I have wireless charging pads all over the house now so I don't think I'll ever charge it any other way.

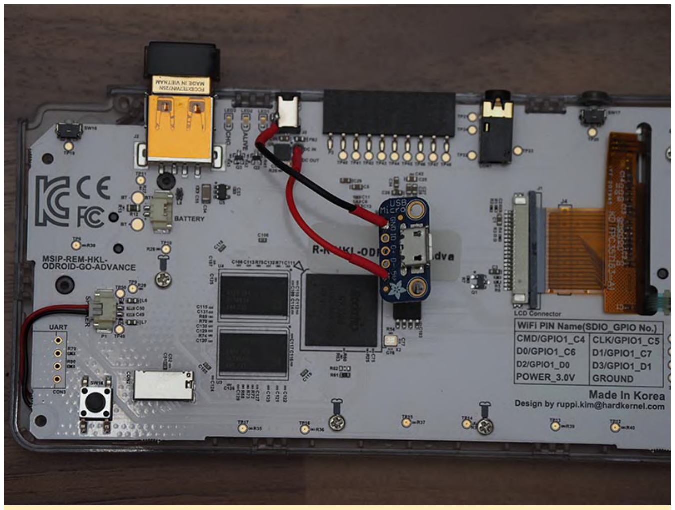


Figure 1 - Closeup of the soldering required to add wireless charging to the ODROID-GO Advance.

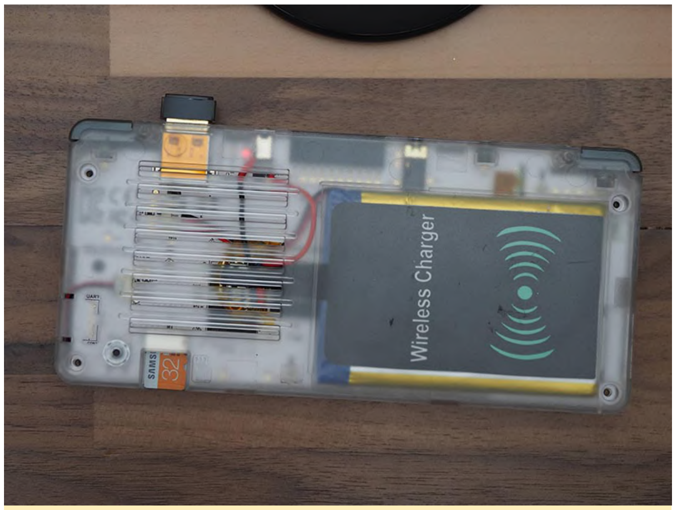


Figure 2 - The wireless charger attached inside the ODROID-GO Advance case.

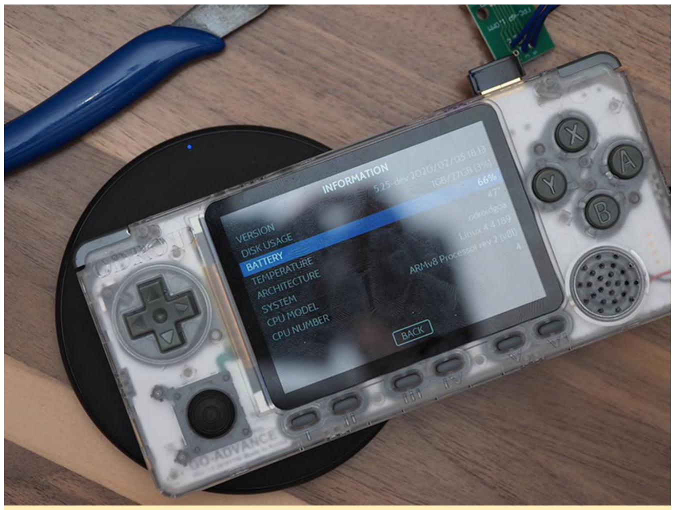


Figure 3 - The wireless charger is now actively charging the ODROID-GO Advance.

For comments, questions, and suggestions, please visit the original article at https://forum.odroid.com/viewtopic.php?f=193&t=38193.

Linux Gaming: GameStation Turbo Advance

April 1, 2020 By Tobias Schaaf Gaming, ODROID-GO Advance, ODROID-GO, Tutorial

ODROID GameStation Turbo Advance is a feature-packed gaming OS for the ODROID-GO Advance utilizing X11 as a graphics backend and attract mode as a frontend for emulators, games and tools.

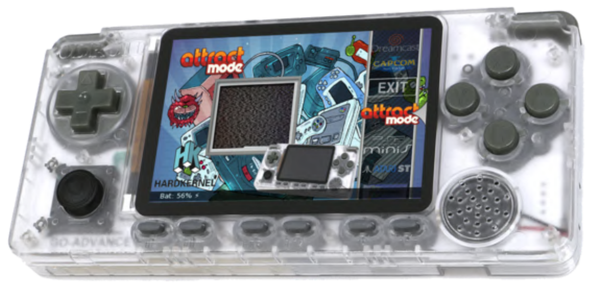


Figure 1 - ODROID-GO Advance running GameStation Turbo Advance

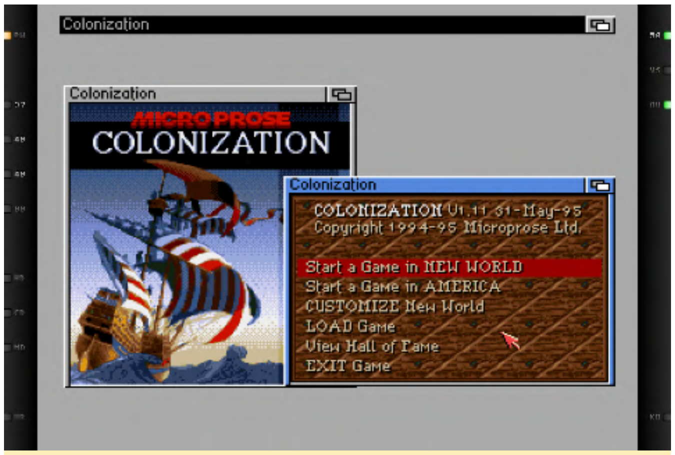


Figure 2 - Play your favorite Amiga games

Figure 3 - Play your favorite Amiga games

Figure 4 - Player your favorite Atari ST games

Figure 5 - Play your favorite ScummVM games

Figure 6 - Player your favorite Nintendo DS games

The goal is to have a shiny and flashy UI, as well as an easy way to integrate new software. It is based on @Meveric's Debian Buster with a lot of modifications to make it work as a gaming image. Please note that I still consider this image to be a 'Work In Progress' as there are still some things I need to improve on, but it is at a point where it can be shared with "the public".

Currently Supported Emulators

- Amstrad CPC
- Atari 2600
- Atari 5200
- Atari 7800
- Atari 8bit
- Atari Lynx
- Atari ST
- Bandai Wonderswan
- Bandai Wonderswan Color
- Capcom Play System
- Capcom Play System I
- Capcom Play System II
- Commodore 64
- Commodore Amiga
- Colecovision
- GCE Vectex
- Magnavox Odyssey 2
- Mattel Intellivision
- Microsoft MSX
- Microsoft MSX2
- NEC SuperGrafx
- NEC Turbografx-16
- NEC Turbografx-CD
- Nintendo 64

- Nintendo DS
- Nintendo Entertainment System
- Nintendo Famicom Disk System
- Nintendo Gameboy
- Nintendo Gameboy Color
- Nintendo Gameboy Advanced
- Nintendo Virtual Boy
- Final Burn Alpha
- Sammy Atomiswave
- ScummVM
- Sega 32x
- Sega CD
- Sega Dreamcast
- Sega Gamegear
- Sega Genesis/Megadrive
- Sega Master System
- Sega Naomi
- Sega SG-1000
- Sharp X68000
- Sinclair ZX 81
- Sinclair ZX Spectrum
- SNK Neo Geo AES
- SNK Neo Geo CD
- SNK Neo Geo Pocket
- SNK Neo Geo Pocket Color
- Sony Playstation
- Sony Playstation Portable
- Sony Playstation Portable Minis
- Super Nintendo Entertainment System

The default BIOS path is /home/odroid/ROMS and you need to place your BIOS files in this folder (if not marked otherwise, please check notes available from the link at the end of the article). It is also the path where you have to place your ROM files, in the respective subfolders. It is available for download at https://oph.mdrjr.net/meveric/images/OG ... GO2.img.xz md5 sha512 sig.

The image utilizes both 32-bit and 64-bit drivers which allows you to run a lot of applications already existing for armhf, even if they are not ported to ARM64, yet.

How To Start

Copy your rom files into /home/odroid/ROMS/ folders that correspond with the system the roms are for (e.g. Super Nintendo Entertainment System goes to /home/odroid/ROMS/SNES/). When you boot up the image you see "Attract Mode Setup" it has two options

- Scrape games and artwork
- Scrape games only

For artwork you need an Internet connection so either have a USB to LAN adapter installed and working correctly. This should work out of the box. Scraped games only will import your games but not download any artwork. Scraping artwork can take a very long time (and disk space) depending on your library. It's not recommended to scrape thousands of games at once, unless you have a couple of hours to spare.

You will see a progress bar while the games and artwork are being imported. This progress bar only shows the overall progress of all emulators not for each individual ROM, which means depending on how many games you import the percentage of the process will not go up until it's finished with the system it's currently importing.

Update Regularly:

Since there's constantly new development for this image and, ODROIDs in general, it is highly recommend to do all updates:

```
$ apt-get update && apt-get upgrade && apt-get
dist-upgrade && apt-get autoremove
```

This will guarantee you get the latest patches and fixes, as well as new emulators when they become available.

Setting up WLAN from the Command Line

Edit the file /etc/network/interfaces or create a new one under /etc/network/interfaces.d/ and add the following line:

```
auto wlan0
iface wlan0 inet dhcp
    wpa-essid
    wpa-psk
```

Main Menu

The Main Menu is powered by Attract Mode and uses the following button layout:

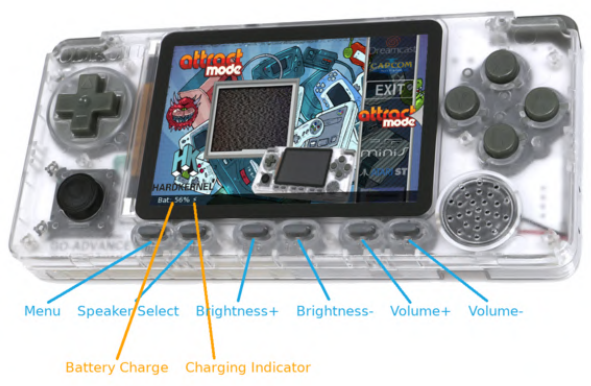


Figure 7 - Button Layout

Although the Buttons work in each sub menu, the output of the button you pressed and the current battery charge is only shown on the main menu. Attract Mode has a screen saver mode, which will activate after about 3 minutes of inactivity. It will show previews of the videos and pictures of the current selection, this means on the main menu it will show videos of all the different systems itself, while in the sub menu of a system it will show previews of the different games. Finally, Speaker Select will cycle through Headphones only, Headphones + Speaker, Speaker only and OFF.

Retroarch

Many of the emulators are powered by retroarch and use the default button layout of retroarch. The special keys are mapped as following:

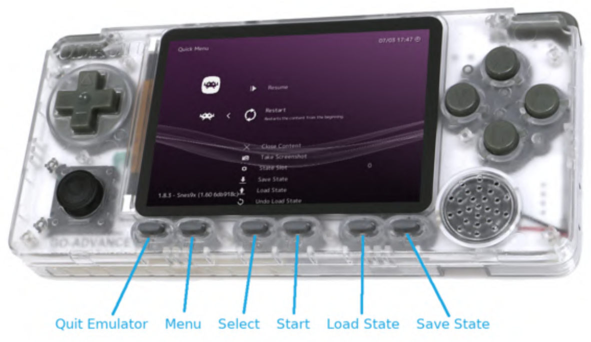


Figure 8 - Special Key Mapping

If you want to do changes on the settings of retroarch, I advise you to exit retroarch if you want to save the changes. If you go back into the game and then exit the emulator, your changes may not be saved.

Amiga Emulator - FS - UAE

For Amiga Emulation I'm using FS-UAE as its user interface is much more controller friendly than other emulators. I also mapped Mouse movement and buttons, as well as special keys for the Amiga. The mapping looks like this:

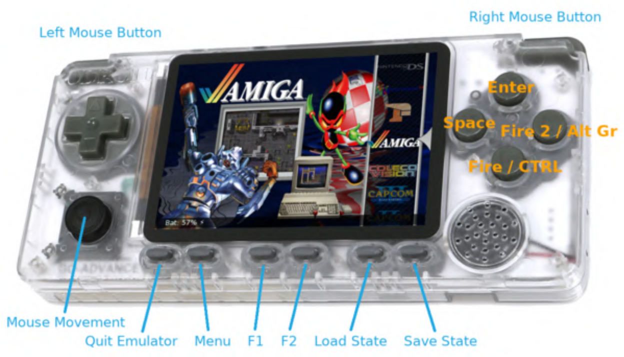


Figure 9 - Amiga Key Mapping

The emulator is currently configured for Amiga 500 compatible games, so your AGA or CD32 games will not work. Since many games come with multiple disks you can zip all adf files into one file. The disks will be inserted into DF0 to DF3 but you can change disks via the menu, even if a game only supports DF0, or has more than just 4 disks.

The Emulator uses separate folders for Kickstart files which can be found under: /home/odroid/Documents/FS-UAE/Kickstarts and under: /home/odroid/.config/fs-uae/ you can find the config files for your different games. You can then, for example, use different Kickstarts if you want, or change memory settings, etc.

Atari ST - Hatari

Atari ST runs on the Hatari Standalone emulator, not a libretro core. It has better performance and much better button mapping. The libretro core works only on a very few number of games. The button layout is as following:

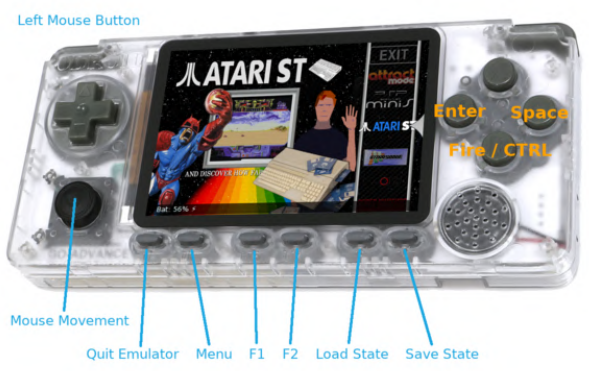


Figure 10 - Atari ST Button Mapping

Some buttons are still free and may be mapped later. There is a way to use the menu to switch disks, but it's quite hard. The menu is not very easy to navigate, therefore I suggest using single disk games, or games that use a harddrive image instead.

Nintendo 64 - Mupen64plus

Mupen64plus standalone emulator is quite a bit faster than the libretro core alternatives, therefore, I use the standalone version here with the RICE video core which is the fastest video core at the moment, even though it's not perfect. The button layout is always a controversy but this is how I mapped the buttons:

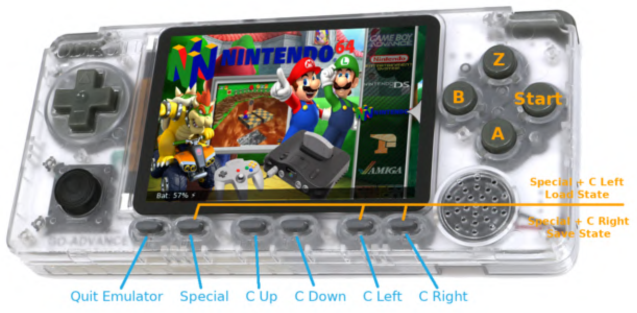


Figure 11 - Nintendo 64 Button Mapping

This is the only emulator where you have to press two buttons for saving and loading save states, sadly there were not enough buttons otherwise. The performance and quality of emulation overall differs a lot between games.

Nintendo DS - DraStic

DraStic by @Exophase is a very fast, but it is a closed source Nintendo DS emulator and therefore a little bit complicated to get to work. The performance is very impressive and you can play NDS very well on the ODROID-GO Advance. The button mapping looks as following:

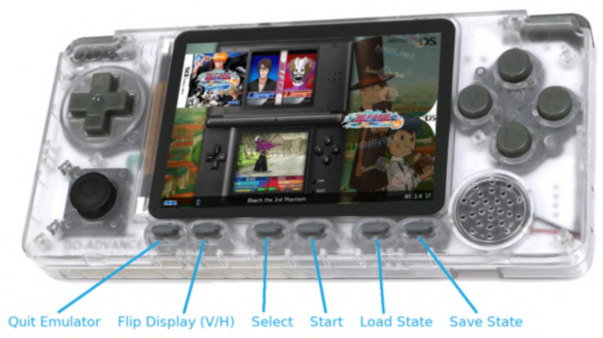


Figure 12 - Nintendo DS Button Mapping

Unfortunately, the NDS (or DSi), has too many functions to map everything, so noise emulation, open and closing the lid are not supported which makes some games unplayable. It seems like every now and then DraStic does not recognize input at all (aside from the mouse pointer). Not sure why, not sure how to avoid it. I found restarting the emulator seems to solve this. Just press the Quit Emulator button and try anew. Make sure you did all system updates before trying this.

Playstation Portable - PPSSPPSDL

PPSSPPSDL is a very well known emulator for PSP and works quite well on the ODROID-GO Advance. It's not as good as other devices, such as the ODROID-XU4 or ODROID-N2, but still good enough for most games. The button layout is as following:

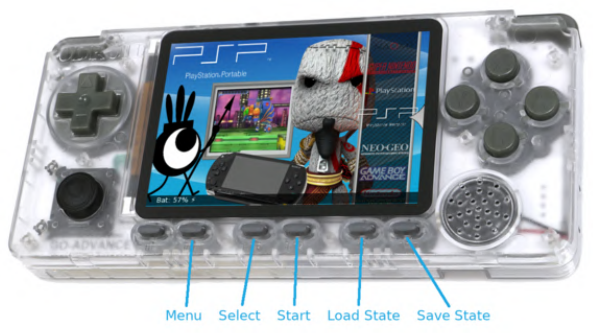


Figure 13 - PSP Button Mapping

ScummVM Hints

The layout for ScummVM games are a little different from other Emulators. In the folder /home/odroid/ROMS/SCUMMVM/ you need to create a folder with the name of the game you want to play, for example, Beneath a Steel Sky (/home/odroid/ROMS/SCUMMVM/Beneath a Steel Sky/). In this folder, you will copy all your game files from the ScummVM game of your choice. The next step is to create a file with the same name and the extension .svm in the same folder "Beneath a Steel Sky.svm" (/home/odroid/ROMS/SCUMMVM/Beneath a Steel Sky/Beneath a Steel Sky.svm). Inside the file you put the ID of the game. In our example this is "sky". You can find the game IDs on the ScummVM Home Page in the column "Game Short Name".

Sharp X68000 Hints

The Sharp X68000 emulator supports single disk images (.dim) and multi disk images .m3u. .m3u is a "playback" list, which just houses the path/name of the .dim files. If you have games that require multiple disks, you should put all disks inside a subfolder inside: /home/odroid/ROMS/X68000/ and create a .m3u file that houses the names of the disk images.

For example: Create a folder /home/odroid/ROMS/X68000/Gradius II

```
$ mkdir -p "/home/odroid/ROMS/X68000/Gradius II"
```

Copy all *.dim files inside this folder

```
/home/odroid/ROMS/X68000/Gradius II/Gradius II
Gofer No Yabou (1992)(Konami)(Disk 1 of 2)(Disk
A).dim
/home/odroid/ROMS/X68000/Gradius II/Gradius II
Gofer No Yabou (1992)(Konami)(Disk 2 of 2)(Disk
B).dim
```

Create a .m3u file that has the path of the ROMS files inside:

```
$ cat "/home/odroid/ROMS/X68000/Gradius II.m3u"
Gradius II/Gradius II Gofer No Yabou (1992)
(Konami)(Disk 1 of 2)(Disk A).dim
Gradius II/Gradius II Gofer No Yabou (1992)
(Konami)(Disk 2 of 2)(Disk B).dim
```

Sony Playstation

Same as Sharp X68000 you can create m3u files and folders for games with multiple CDs. You can switch the CDs from the retroarch menu. For more information please see the original article, located at https://forum.odroid.com/viewtopic.php?f=193&t=38177.