Linux Gaming • RetroArch • Meet An Odroidian • NEMS Linux: Part 3

ODROID Magazine

Year Six
Issue #60
Dec 2018

YOUR PORTABLE ODROID IN A WAY YOU ALWAYS DREAMED:

To Boldy go:

A Tricorder

ODROID-GO

CODING CAMP:
MAKE A PORTABLE HANDHELD WEATHER STATION

COMMODORE 64:
NOW FOR YOU, EMULATE THE FIRST TRUE HOME COMPUTER

Linux Gaming – DOSBox, an x86 DOS Emulator: Play Your Original DOS Games in HD

December 1, 2018

DOSBox is an x86 DOS Emulator that not only emulates the x86 architecture, but also emulates a common 1990s-era DOS environment. With DOSBox, you can replay your old games and play them on modern hardware, since there are many interesting and legendary DOS applications that aren't available for Windows or ▶

Coding Camp – Part 9: Make A Portable Handheld Weather Station

December 1, 2018

Let us learn how to access various weather data and share it with your mobile devices via WiFi connectivity.

The ODROID-GO Tricorder Project

December 1, 2018

For those of you who do not know what a Tricorder is, allow me to explain: In the newer Star Trek series, characters often carry a mobile device used for things like measuring rips in the space-time continuum and declaring "He's dead, Jim."

Building a Commodore 64 Emulator

December 1, 2018

This emulator allows one play to games designed for the Commodore 64 8-bit system.

Coding Camp – Part 10: Measure the distance with Ultrasonic

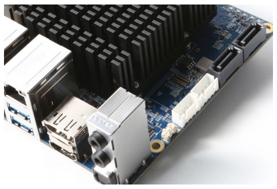
December 1, 2018

Let us learn how to use GPIO output, IRQ input and system timer with a Ultrasonic distance measuring module

Introducing NEMS Linux: Part 3 – Configuring Service Monitors on NEMS Linux

December 1, 2018

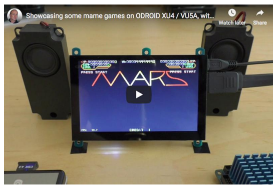
The intention with these articles has been to introduce you to NEMS Linux in such a way as to arm you with useful knowledge that gets you up and running immediately. These aren't intended to appear as documentation, but rather a technical article that gives you ideas as to how ▶



ODROID-H2 Part 2: Bios Features and Remote Access

December 1, 2018

Like a generic PC, the ODROID-H2 has a soldered 8MiB BIOS Flash ROM on the board. It meets the UEFI Specification 2.6 and the PXE boot requirement. However, Intel UEFI firmware doesn't support CSM version 2.0 for legacy OS booting such as DOS, XP, Windows 7, and so on.



Building RetroArch

December 1, 2018

If you are looking for a frontend to game emulators, you can try RetroArch. It has been ported to the ODROID-XU4 family of Single Board Computers (SBC's). You can follow the steps below to install and use it on your system.

Meet An ODROIDian: Kamots Tech

December 1, 2018

I live in Florida (aka the Sunshine State), where I was born and raised. I have always lived in Florida because it is warm, there's so much to do, and the IT industry has been growing steadily with a lot of promise on the horizon. I went to college for ▶️

Linux Gaming – DOSBox, an x86 DOS Emulator: Play Your Original DOS Games in HD

December 1, 2018 By Tobias Schaaf Gaming, ODROID-C0, ODROID-C1+, ODROID-C2, ODROID-XU4

DOSBox is an x86 DOS Emulator that not only emulates the x86 architecture, but also emulates a common 1990s-era DOS environment. With DOSBox, you can replay your old games and play them on modern hardware, since there are many interesting and legendary DOS applications that aren't available for Windows or Linux.

DOSBox is very stressful on many computers, since you normally need a high-end PC to emulate a 486 at 33MHz. Since the ODROID uses a completely different architecture (ARM vs X86), it has even more work to do during emulation. Despite its complexity and multiple layers, DOSBox runs surprisingly well on the ODROID platform.

Some time ago, I compiled an ARMv7-optimized version of DOSBox which appeared to be running faster than the stock DOSBox version that comes with the official distribution. I took some time to compare these versions and find out exactly what is improved by using an ARMv7-optimized build.

Below you will find a series of side-by-side tests that highlight the differences between the generic build of DOSBox, and a build that is specifically compiled for ARM. The custom build of DOSBox for ARMv7 may be downloaded from my repository at http://bit.ly/1DhCv6l.

Configuration

Configuring DOSBox can sometimes be difficult. While most games run fine with the basic settings, others only run with a very specific configuration, so I chose a set of values that worked best for the original version of the game Quake, since it's very demanding on the hardware.

What's remarkable about Quake is that the game itself is in 3D without requiring a graphical desktop environment. In contrast to games like “Duke Nukem 3D” ,which contains some 3D objects and use 2D sprites in many situations, Quake was already using 3D models, similar to the models used in later games on Windows, which was very impressive for that time.

There was no easy way to find the right settings, and after a period of experimentaiton, I ended up with the following results, with frameskip and aspect ratio turned off:

```
core=dynamic
cputype=pentium_slow
cycles=fixed 32000
cycleup=500
cycledown=300
memsize=32
scaler=normal3x
```

Dynamic cores should be used for any value of fixed cycles over 20,000. Pentium_slow is the CPU with the most features, and I set the cycles to 32,000, which is very high. Some test programs reported it to be a 1285 MHz fast Pentium CPU. I chose such a high number because of Quake, since at 32,000 cycles it gives the most fluid experience on both DOSBox versions.

Tests

After performing a variety of tests, I discovered that it was actually hard to find some good benchmarks. I remembered some benchmarking applications from back when DOS was popular, but they were hard to find. However, I did find a test environment for performing different benchmarks under DOS called DOS Benchmark, which is available for download from http://bit.ly/1ttzaRR.

DOS Benchmark offers CPU, GPU, and memory tests, as well as demo versions of the games Doom and Quake for benchmarking the environment. I tried to run every test available, but not all of them were working, although I did find a few that performed nicely. For instance, I found a test with a spinning 3D cube running in DOS, which has some great visuals, and ran relatively fast on the ODROID.

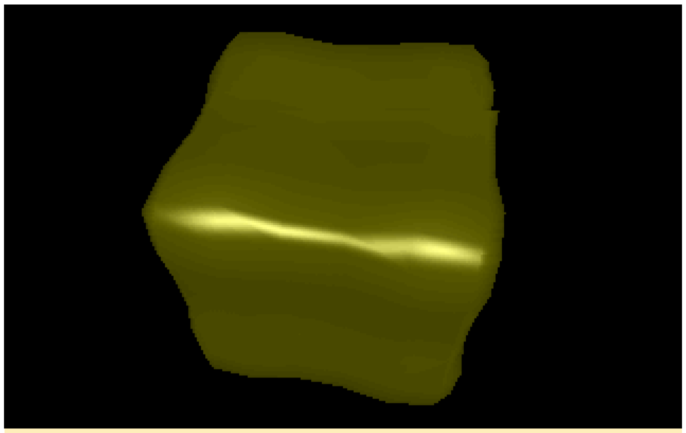
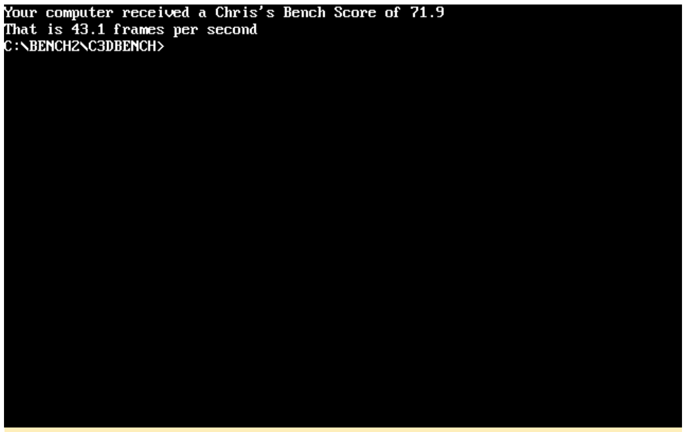
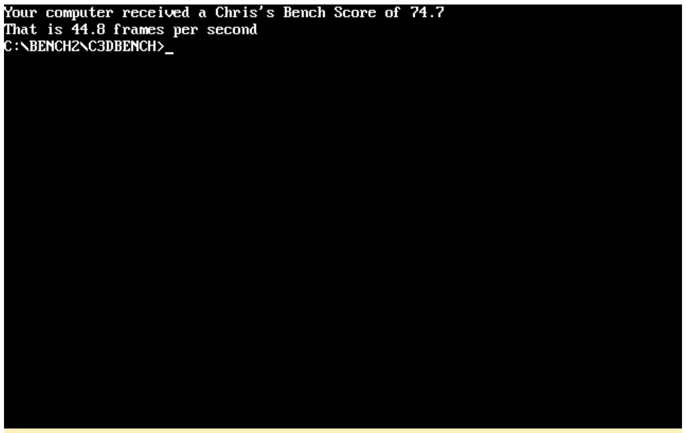


Figure 1 – Spinning cube under DOS


```
Your computer received a Chris's Bench Score of 71.9
That is 43.1 frames per second
C:\BENCH2\C3DBENCH>
```
Figure 2 – Stock Debian version of DOSBox


```
Your computer received a Chris's Bench Score of 74.7
That is 44.8 frames per second
C:\BENCH2\C3DBENCH>_
```
Figure 3 – ARMv7-optimized version of DOSBox

3DBench test

The ARMv7-a optimized version was nearly 17% faster in this test. Unfortunately, that test is not very reliable if you change the CPU cycles as I did. I was able to achieve results with over 200 FPS with values like 100,000 CPU cycles, but even with these high numbers, the emulator was far from working better or even faster. I could see that the video output was lagging and skipping frames, but the test still got high scores.

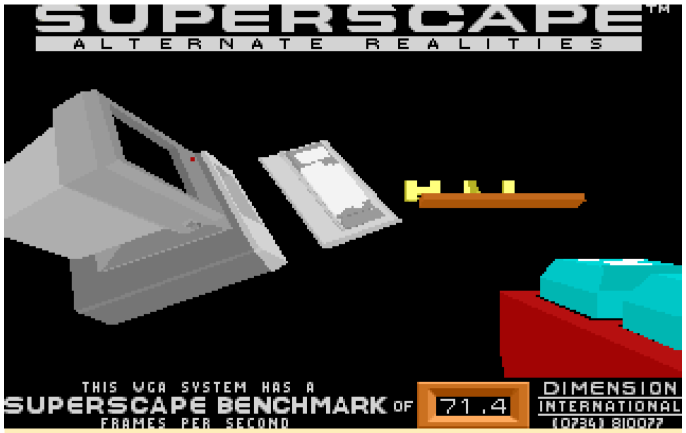


Figure 4 – 3D Bench test using stock build

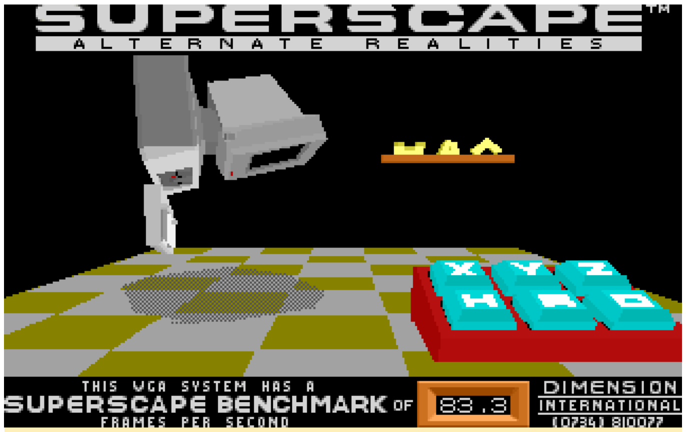
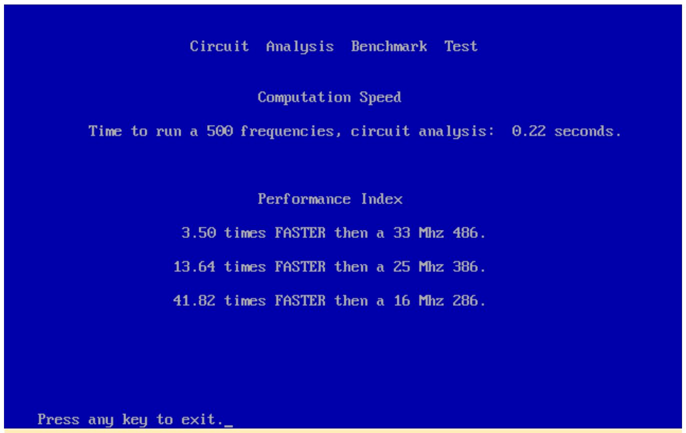


Figure 5 – 3D Bench test showing difference in results using ARM build

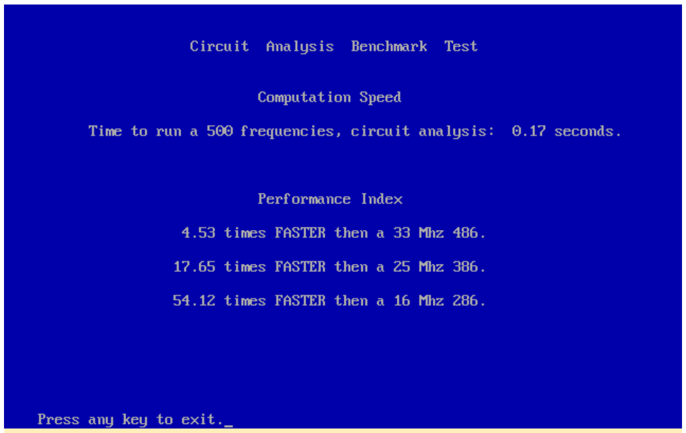
Benchmark

The CPU tests were showing that the ARMv7-a optimized version performs somewhat better. An improvement of around 30% was common when it came to CPU computing comparisons.



```
Circuit  Analysis  Benchmark  Test

Computation Speed

Time to run a 500 frequencies, circuit analysis:  0.22 seconds.


Performance Index

3.50 times FASTER then a 33 Mhz 486.

13.64 times FASTER then a 25 Mhz 386.

41.82 times FASTER then a 16 Mhz 286.



Press any key to exit._
```

Figure 6 – CPU benchmark using stock build



```
Circuit  Analysis  Benchmark  Test

Computation Speed

Time to run a 500 frequencies, circuit analysis:  0.17 seconds.


Performance Index

4.53 times FASTER then a 33 Mhz 486.

17.65 times FASTER then a 25 Mhz 386.

54.12 times FASTER then a 16 Mhz 286.



Press any key to exit._
```

Figure 7 – CPU benchmark using ARM build, which is clearly faster

Memory issues

While some benchmarks worked better in the ARM version, I saw some major issues in some tests when it came to the ARMv7-a optimized version. Some tests didn't even run on the ARMv7-a optimized version of DOSBox, or causes strange behavior. Only the stock Debian version was running 100% of the tests correctly. (Figure 8 – CACHECHK was only working on the stock Debian version of DOSBox, and properly identified the CPU)

For example, there was a memory test which used blocks of different sizes and did some operations on them so that in the end the different blocks added up to 24MB in total. It operated on 384 x 64KB blocks and gave a result on how fast the memory did the computing. The same test on the optimized version resulted in very different results. Not only did the ARM test took approximately 10 times as long to run, the values were completely inaccurate. Instead of 24MB in results it added up blocks of 512 MB and more at a ridiculous speed.

Some tests went so high, that it went out of scale and resulted in either a negative speed or with high exponents calculating ten thousands of megabytes per second. Other tests didn't start at all, or caused the emulator to freeze.

Testing tools

I tried some other testing tools for benchmarking the graphics performance of the system, like the spinning cube and VideoDOS, which sometimes had very odd results. Because the graphical tests are just benchmarks, and don't directly relate to gameplay responsiveness, I did some hands-on testing with some of my favorite games as well.

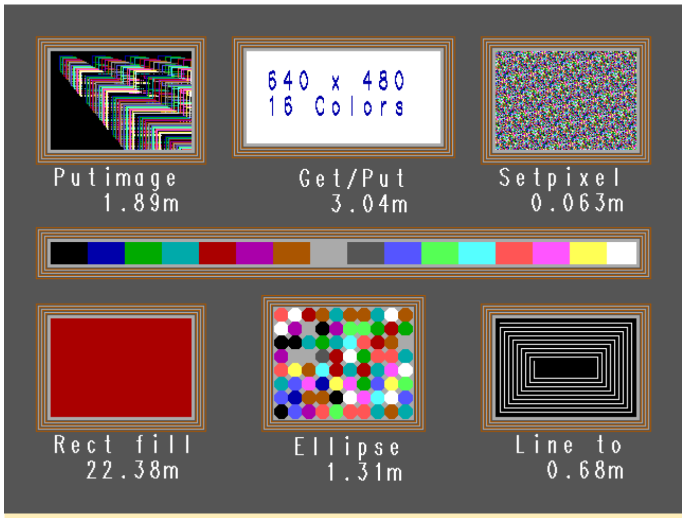


Figure 9 – Graphical test on the stock Debian build of DOSBox

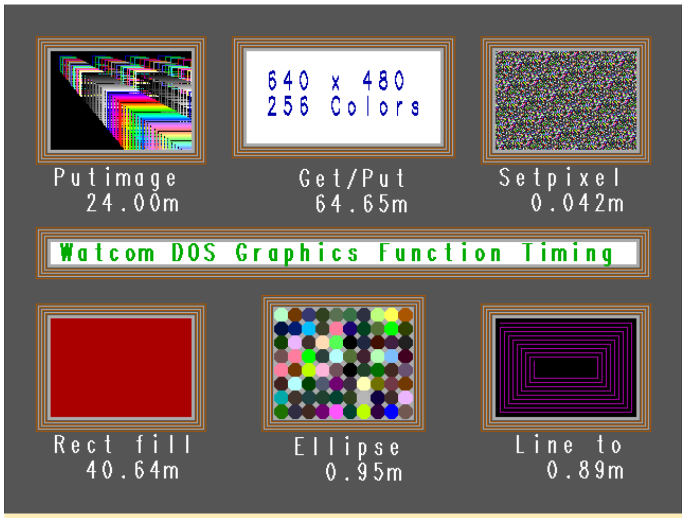
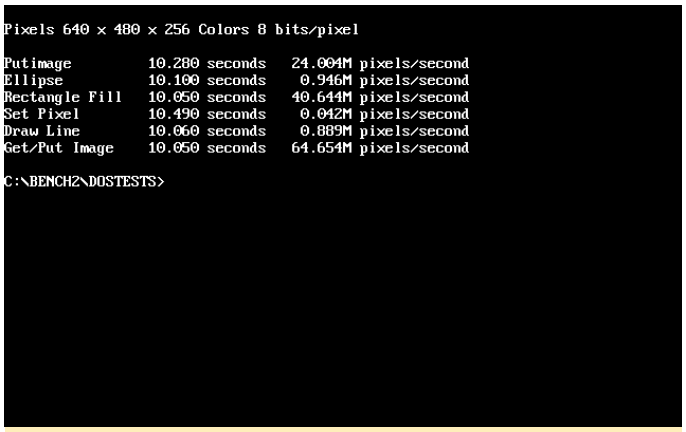
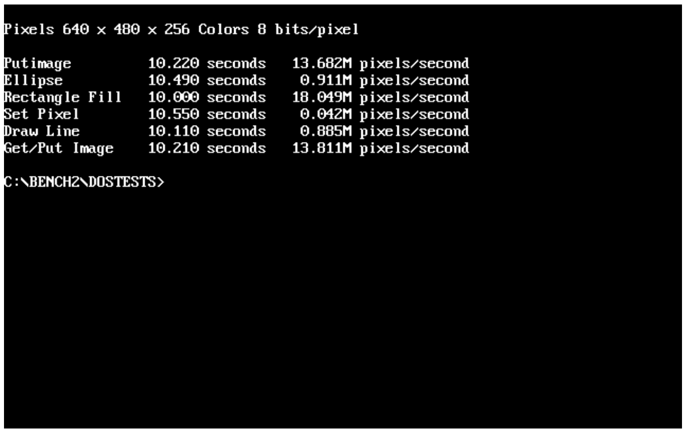


Figure 10 – Graphical test on the ARM build of DOSBox. This graphic tests gave odd results: some tests seemed to run faster with more colors and in higher resolutions, while others seemed normal)



```
Pixels 640 x 480 x 256 Colors 8 bits/pixel

Putimage        10.280 seconds   24.004M pixels/second
Ellipse         10.100 seconds    0.946M pixels/second
Rectangle Fill  10.050 seconds   40.644M pixels/second
Set Pixel       10.490 seconds    0.042M pixels/second
Draw Line       10.060 seconds    0.889M pixels/second
Get/Put Image   10.050 seconds   64.654M pixels/second

C:\BENCH2\DOSTESTS>
```

Figures 11 and 12 – Results of VideoDOS of the ARM optimized version (top) and the stock Debian version (bottom)



```
Pixels 640 x 480 x 256 Colors 8 bits/pixel

Putimage        10.220 seconds   13.682M pixels/second
Ellipse         10.490 seconds    0.911M pixels/second
Rectangle Fill  10.000 seconds   18.049M pixels/second
Set Pixel       10.550 seconds    0.042M pixels/second
Draw Line       10.110 seconds    0.885M pixels/second
Get/Put Image   10.210 seconds   13.811M pixels/second

C:\BENCH2\DOSTESTS>
```

Games

The benchmark package included two games, Doom and Quake, since both were very commonly played during the golden age of DOS, and offered some nice benchmark features in demo mode. However, the benchmark build into Doom did not work correctly, and claimed to nearly always be running at nearly full speed, although it was far from it. (Figure 13 – The Doom DOS version is playable but not very smooth on DOSBox, but performs much better as a native Linux port)

Instead of using the built-in benchmarks, I did my own testing and compared the time that it took the games to do a full demo run. The results were very surprising for me: Demo 3 took about 108 sec to complete a full demo run on the ARMv7-a optimized version of DOSBox, and on the stock Debian version of DOSBox it took 156 sec instead. That's a nearly 45% increase in speed for the ARM version.

Even more dramatically, you could see the difference when playing Quake. A demo run for Demo 3 took 147 sec on the optimized version and 248 sec on the stock Debian version, which is approximately 70% faster! After all the benchmarking, I wanted to see how well the emulator performs in a real gaming experience, and soon found out that the settings that I had originally chosen did not work well for any other games, so I changed the settings again and ran a couple of test games. After I tuned down the cycles to 6,000 instead of 32,000, Dune 2 was running perfectly fine, with nice, smooth gameplay. The sounds, music and voices were all good, and I had no other issues.

I also tried a couple more demanding games, such as Prisoner of Ice, which is a very nice adventure game with some movie cut-scenes and an option to either run in 320×240 resolution or in 640×480. The last one even offered some other features such as better fonts. Both versions were running fine on DOSBox. I also found the same superior performance while playing Space Quest 6.

Results

The ARMv7-a optimized version runs significantly better than the stock Debian version of DOSBox. If I would have to estimate a number, then the optimized version is, on average, 10 to 15% faster than the version from the Debian repository. Sometimes, it was even far faster than that, such when running Quake.

The faster results seem to be related to some math optimizations inside the emulator, which may also create issues as a side effect, especially with memory operations. This, in turn, may cause glitches in some games, or prevent them from running properly. Besides that, the ARM optimized version is the better version when it comes to speed.

From my previous testing, I can say it's even fast enough to handle Windows 3.11 or even Windows 95. Most games should run nice on both emulators but run are just a little better on the ARM optimized version.

Additional configuration

When I was done with the testing played some games, I changed my settings to the following options, which I found worked well with many games:

```
core=auto or dynamic
cputype=auto
cycles=fixed 3000
memsize=31
```

I also found that DOSBox is able to use glshim together with its opengl renderer using the output option:

```
output=opengl
```

Finally, I changed the sdl settings:

```
fullscreen=true
fulldouble=true
fullresolution=1280x720
windowresolution=original
output=opengl
```

These options start the game in fullscreen mode, and when used together with LD_LIBRARY_PATH=/usr/local/lib/, you can run the emulator with OpenGL support.

Other games

As you can see from the chart below, games vary a lot in playability, and there is no single setting file that works with all games. I also found that the “auto” mode on cycles does not really work well. The 100% speed it uses on heavy games is often worse than using a fixed cycle value.

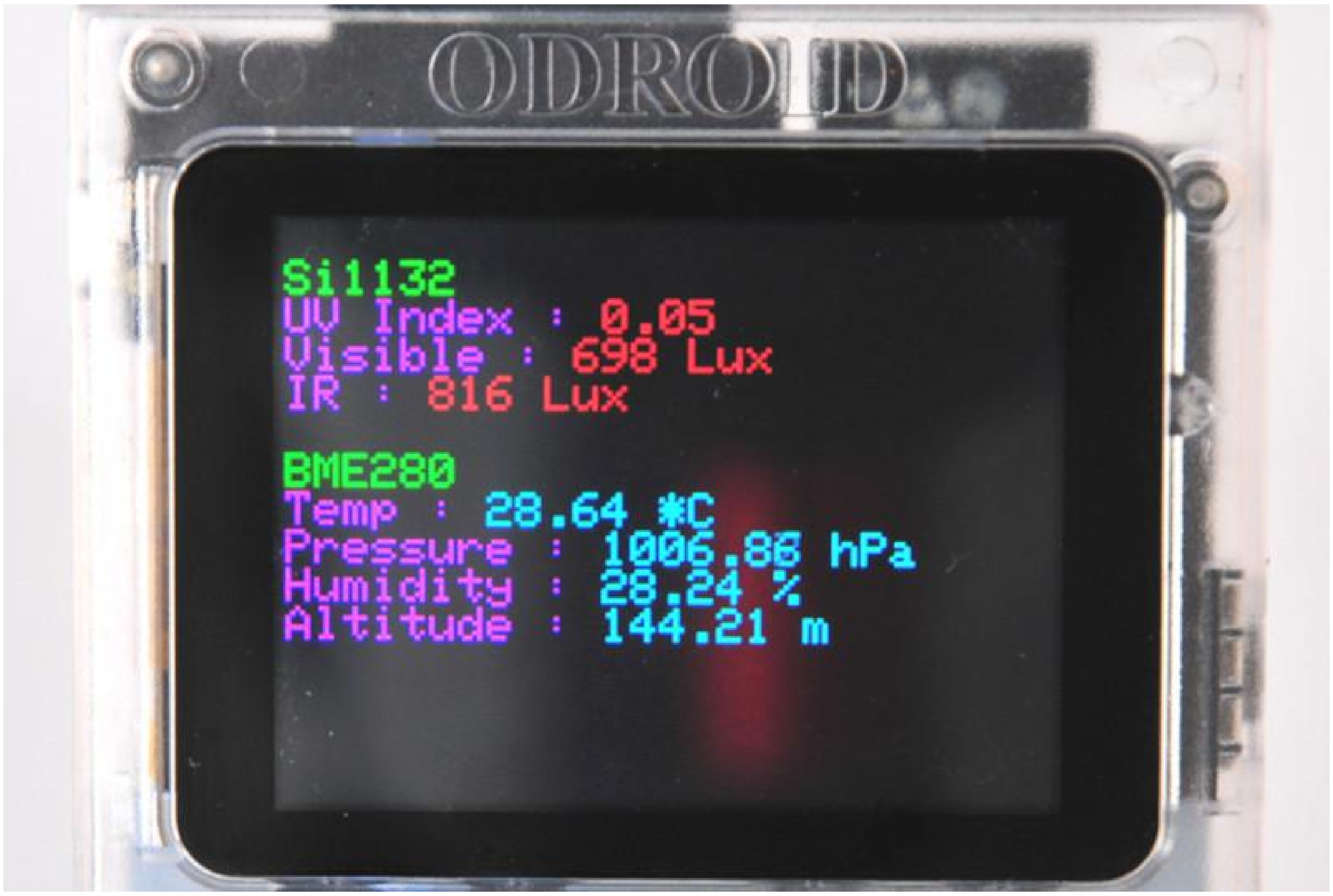
When using DOSBox for your own games, I suggest starting at a cycle value of 3,000, and working your way up until the game starts getting slow again, then taking a few steps back. This should result in the optimum playability for your favorite DOS games.

Game	Cycles	Infos	Comments
Sid Meier's Colonization	1,500-3,000	Game runs best with rather low cycles. Besides that, it's running very fine with no issues or sound drops. However, the intro on the first game start takes a long time to play through.	
Shadow Warrior	15,000-20,000	The game is laggy, and not playable	
Terry Pratchett's Discworld 1	3,000-6,000	Game ran fine without any issues	
Syndicate	6,000-10,000	Game ran fine without	Does not run with

		any issues	glshim
Wing Commander I	2,000-4,000	Game ran fine without any issues. In my opinion, the Amiga version has a much better soundtrack	You should use a 3x scaler here
Prisoner of Ice (640×480)	2,000-8,000	Game ran fine with only a minor issue with the sound cracking occasionally	
Space Quest 6	~12,000	Game mostly runs at full speed, but has some slight stuttering in the music, and the text is too fast	
Dune 2	3,000	Game seems a little slow but generally good and without issues	
XCom Series	1,000-15,000	Works well with only slight speed issues	
Dark Legions	~20,000	Works well with only slight speed issues	

Coding Camp – Part 9: Make A Portable Handheld Weather Station

December 1, 2018 By Justin Lee ODROID-GO, Tinkering, Tutorial



```
Si1132
UV Index : 0.05
Visible : 698 Lux
IR : 816 Lux

BME280
Temp : 28.64 *C
Pressure : 1006.86 hPa
Humidity : 28.24 %
Altitude : 144.21 m
```

Let us learn how to access various weather data and share it with your mobile devices via WiFi connectivity. Note that the Weather Board 2 is additionally required (https://wiki.odroid.com/odroid_go/arduino/30_weather_station).

Before we begin, this Coding Camp will use a web browser to show weather information. To make things go smoothly, make sure you have the following two bullet points meet. Make sure that you've followed the Arduino setup guide.

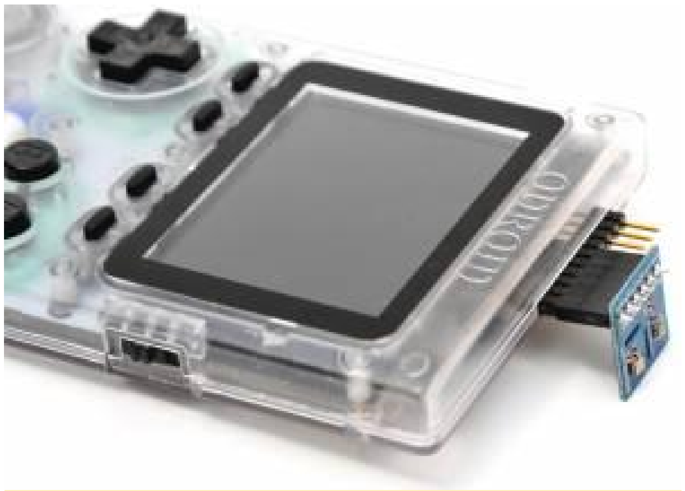


Figure 1 – You can have a portable Weather station in your hand

Requirements

Make sure that you have these products:

- ODROID-GO
- Weather board 2

- A MicroUSB cable

Setup the development environment for Arduino on your system. Before proceeding with this guide, attach Weather board 2 to ODROID-GO and connect it to the PC via micro USB cable.

Setup SPIFFS

SPIFFS stands for SPI Flash File System. You can visit **https://github.com/me-no-dev/arduino-esp32fs-plugin** to see full documentation about SPIFFS. ODROID-GO has a small (but enough to use) flash memory which you can upload data by using this tool. Download a compressed file from **this link (ESP32FS-v0.1)**, then extract the ESP32FS directory to one of the following directories, depending on your operating system:

- Windows: %USERPROFILE%DocumentsArduino ools
- Linux: ~/Arduino/tools

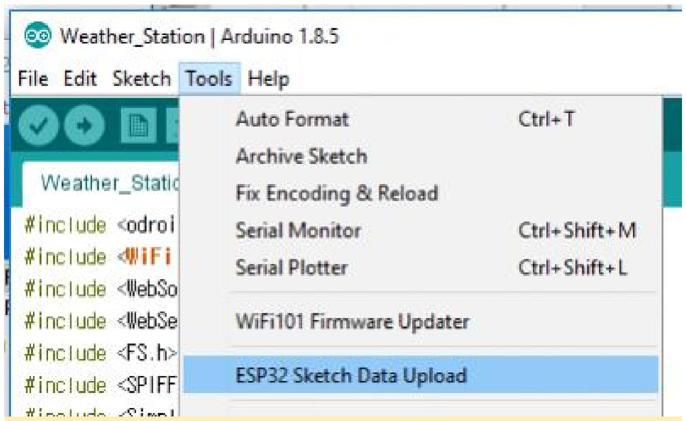


Figure 2 – Arduino Tools directory

(Figure 2 – Arduino Tools directory)

Create the tools directory before extracting if it doesn't exist. Open Arduino IDE, and you can see the Tools → ESP32 Sketch Data Upload menu.

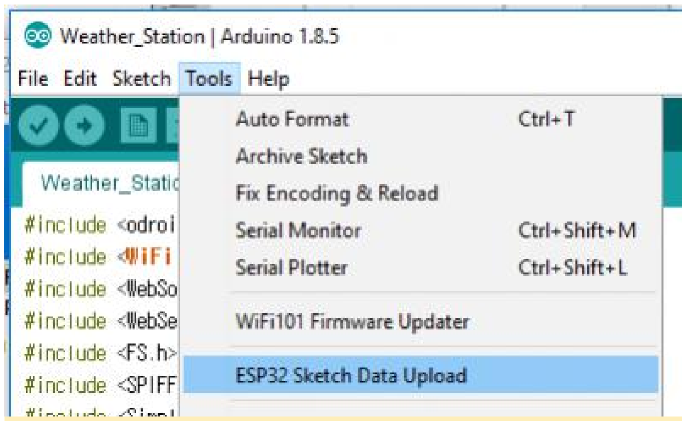


Figure 3 – upload the sketch

(Figure 3 – upload the sketch)

Import the sample to the IDE

Click the Files → Examples → ODROID-GO → Applications → Weather_Station menu to import the Weather station example.

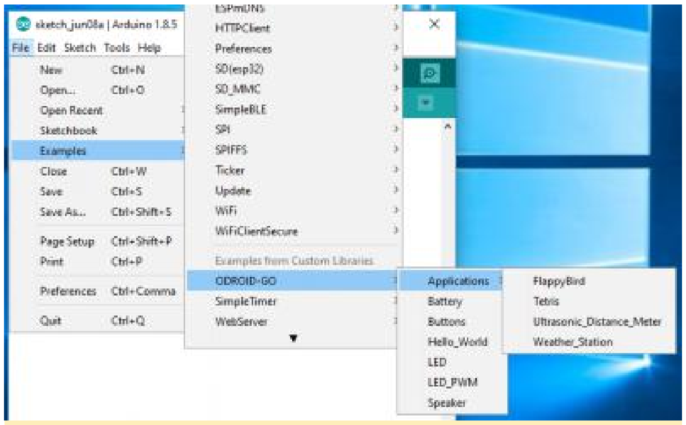
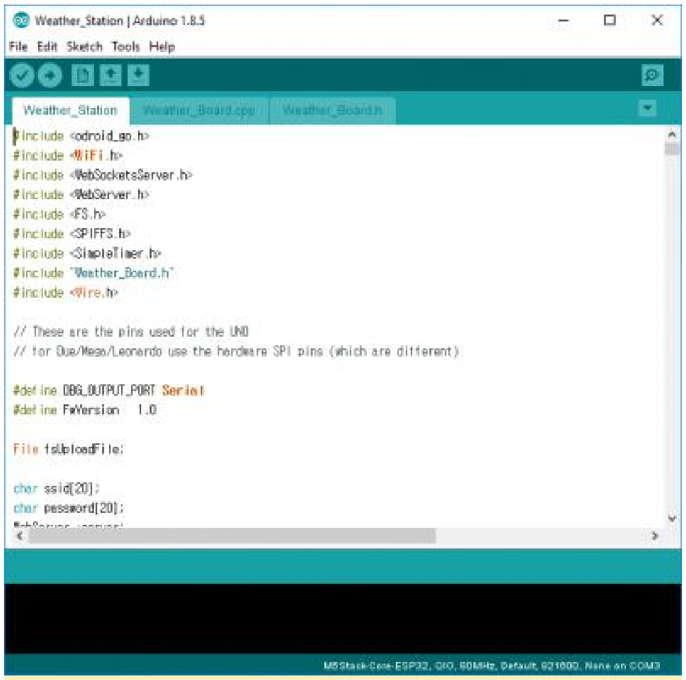


Figure 4 – Arduino Application Selection

Then, you will see a new window with the example code appear.



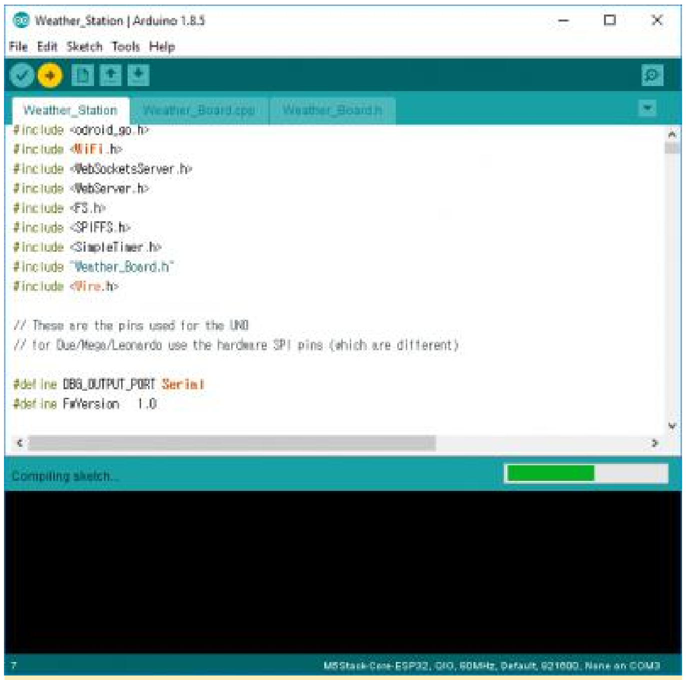
```
#include <odroid_go.h>
#include <WiFi.h>
#include <WebSocketsServer.h>
#include <WebServer.h>
#include <FS.h>
#include <SPIFFS.h>
#include <SimpleTimer.h>
#include "Weather_Board.h"
#include <Wire.h>

// These are the pins used for the UNO
// for Due/Mega/Leonardo use the hardware SPI pins (which are different)

#define DBG_OUTPUT_PORT Serial
#define FwVersion   1.0

File fsUploadFile;

char ssid[20];
char password[20];
```

M5Stack-Core-ESP32, QIO, 80MHz, Default, 921600, None on COM3

Figure 5 – Weather Code

Compile and upload the binary

This guide assume that the port number is COM3. It might be different from yours. Verify and compile the sketch by clicking Sketch → Verify/Compile menu, or by pressing the CTRL-R shortcut.



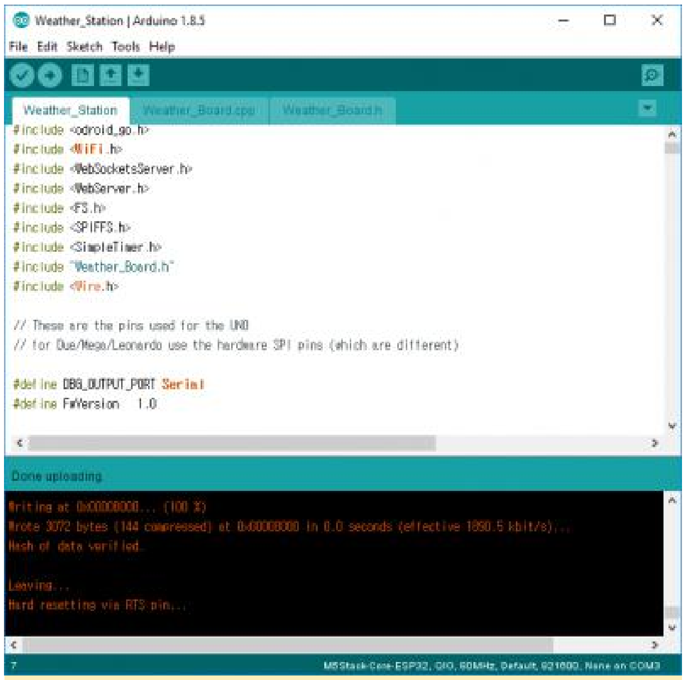
```
#include <odroid_go.h>
#include <WiFi.h>
#include <WebSocketsServer.h>
#include <WebServer.h>
#include <FS.h>
#include <SPIFFS.h>
#include <SimpleTimer.h>
#include "Weather_Board.h"
#include <Wire.h>

// These are the pins used for the UNO
// for Due/Mega/Leonardo use the hardware SPI pins (which are different)

#define DBG_OUTPUT_PORT Serial
#define FwVersion   1.0
```

Compiling sketch...

M5Stack-Core-ESP32, QIO, 80MHz, Default, 921600, None on COM3

Figure 6 – Compiling Sketch

If the compiling completes without any issue, upload the compiled binary by clicking Sketch → Upload or by pressing the CTRL-U shortcut.



```
#include <odroid_go.h>
#include <WiFi.h>
#include <WebSocketsServer.h>
#include <WebServer.h>
#include <FS.h>
#include <SPIFFS.h>
#include <SimpleTimer.h>
#include "Weather_Board.h"
#include <Wire.h>

// These are the pins used for the UNO
// for Due/Mega/Leonardo use the hardware SPI pins (which are different)

#define DBG_OUTPUT_PORT Serial
#define FwVersion   1.0
```

Done uploading.

```
Writing at 0x00000000... (100 %)
Wrote 3072 bytes (144 compressed) at 0x00008000 in 0.0 seconds (effective 1890.5 kbit/s)...
Hash of data verified.

Leaving...
Hard resetting via RTS pin...
```

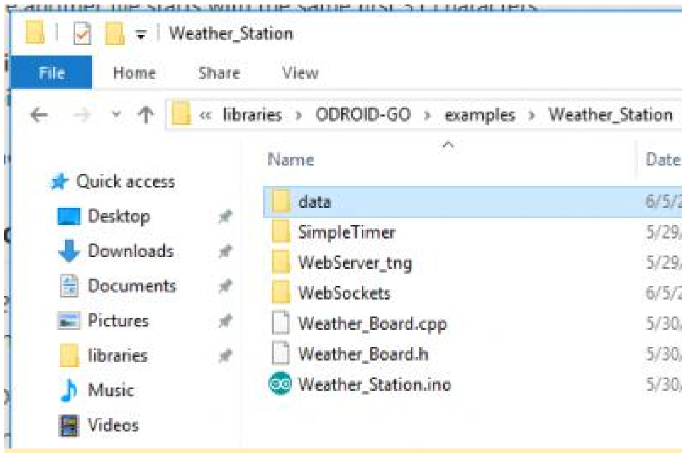
M5Stack-Core-ESP32, QIO, 80MHz, Default, 921600, None on COM3

Figure 7 – Uploading the Sketch

You can omit the compile process since its done automatically when you just upload without doing a compile before. You will know the uploading is complete when the message: "Hard resetting via RTS pin..." appears.

Upload the data

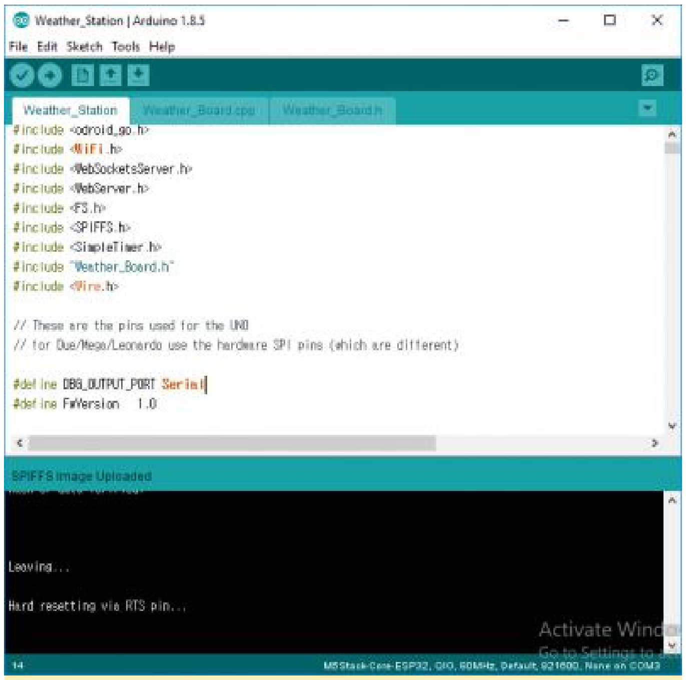
This example has a web server program to serve measurements via a web page. To see that page, you have to upload the web page data to the ODROID-GO's flash memory. SPIFFS lets you do that. If you click the Tools → ESP8266 Sketch Data Upload menu, SPIFFS utility will find the data directory in the current library and send it.



Name | Date
data | 6/5/2
SimpleTimer | 5/29,
WebServer_tng | 5/29,
WebSockets | 6/5/2
Weather_Board.cpp | 5/30,
Weather_Board.h | 5/30,
Weather_Station.ino | 5/30,

Figure 8 – data folder

Click the menu to upload.



```
#include <odroid_go.h>
#include <WiFi.h>
#include <WebSocketsServer.h>
#include <WebServer.h>
#include <FS.h>
#include <SPIFFS.h>
#include <SimpleTimer.h>
#include "Weather_Board.h"
#include <Wire.h>

// These are the pins used for the UNO
// for Due/Mega/Leonardo use the hardware SPI pins (which are different)

#define DBG_OUTPUT_PORT Serial
#define FwVersion   1.0
```

SPIFFS image Uploaded

```
Leaving...

Hard resetting via RTS pin...
```

```
14                    M5Stack-Core-ESP32, QIO, 80MHz, Default, 921600, None on COM3
```

Figure 9 – Uploading SPIFFS image

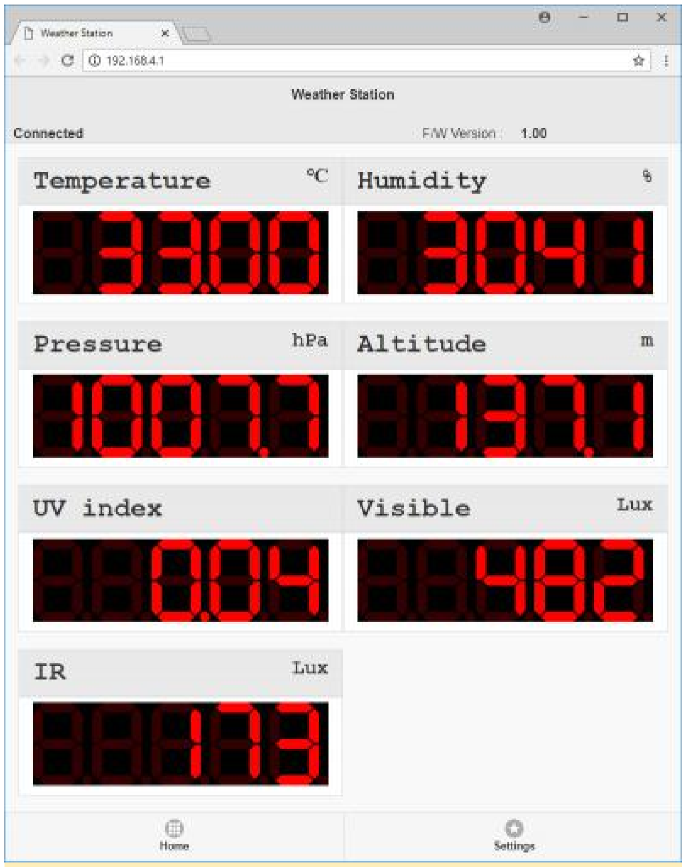
The uploading is completed when the message “Hard resetting via RTS pin...“ is displayed.

Testing

After the upload completes, the ODROID-GO reboots automatically. The screen showing each data measurement appears on the ODROID-GO, and after few seconds, the blue LED in the middle of the board turns on.

Visit with your device – PC / mobile

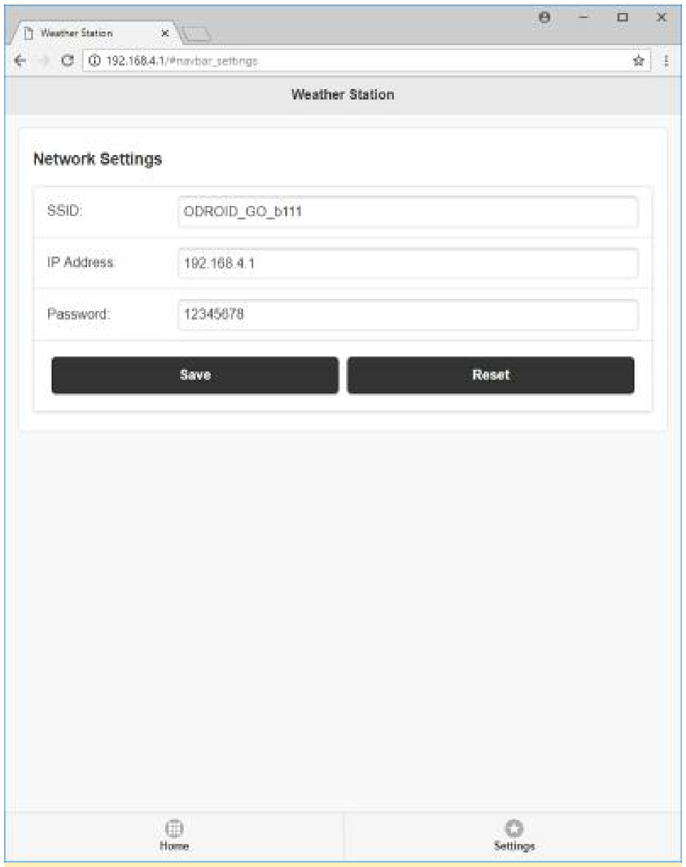
The blue LED indicates that the web server on the board is ready so that you can connect to it and read the data from the ODROID-GO’s access point. Find the Wifi AP named ODROID_GO_**** and connect to it (the default password is 12345678). Open a web browser and navigate to 192.168.4.1. This IP address is set by default. You will see the web GUI showing each measurement, and the blue LED now keeps blinking after the socket communication starts. Using the Network Settings web page, you can set the Wifi configurations such as SSID, IP address, and password.



Connected F/W Version : 1.00

Temperature °C 33.00
Humidity % 30.41
Pressure hPa 1007.7
Altitude m 137.1
UV index 0.04
Visible Lux 482
IR Lux 173

Figure 10 – Weather Station Webpage



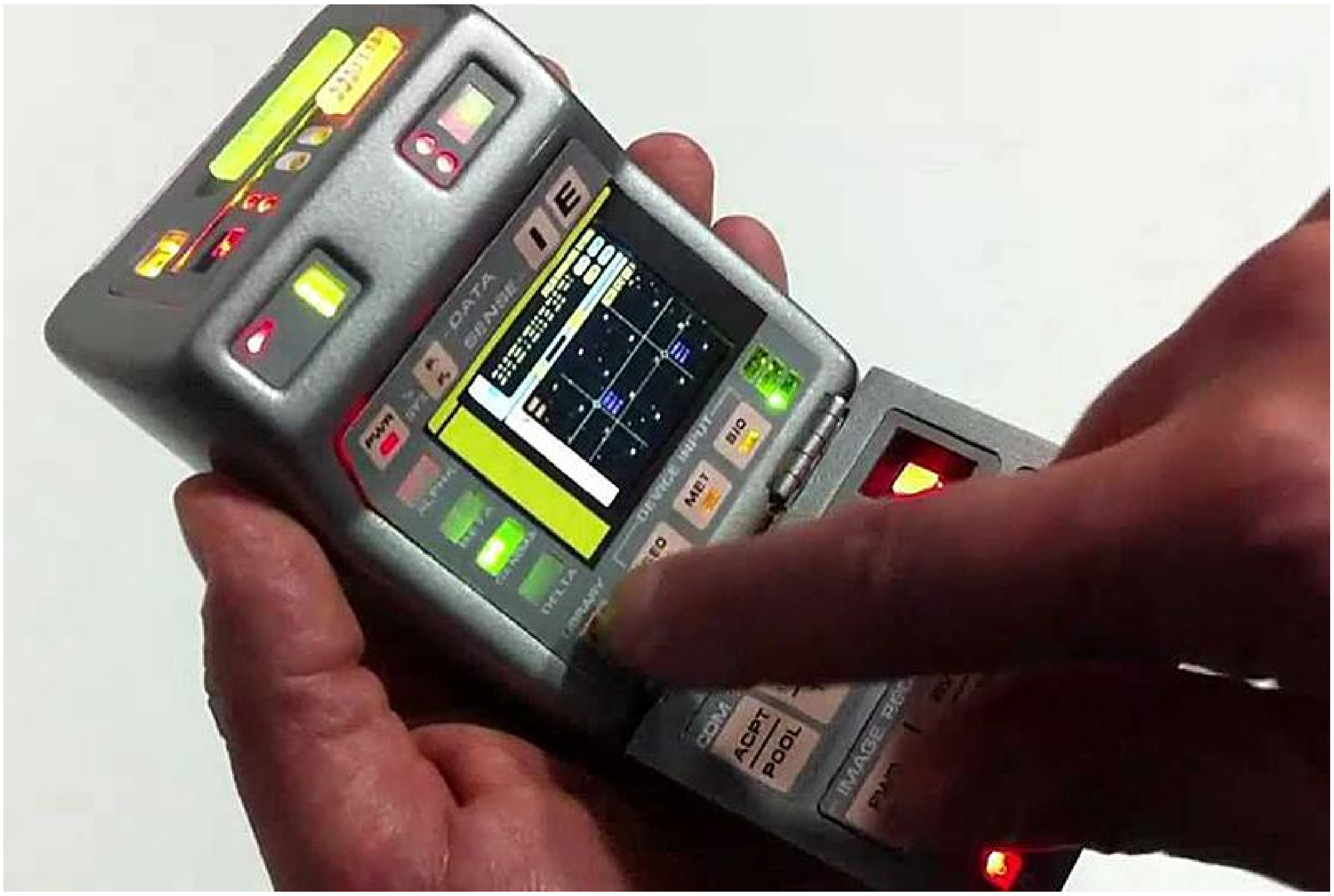
Network Settings

SSID: ODROID_GO_b111
IP Address: 192.168.4.1
Password: 12345678

Figure 11 – Network Settings Webpage

The ODROID-GO Tricorder Project

December 1, 2018 By Volker Raum ODROID-GO, Tinkering



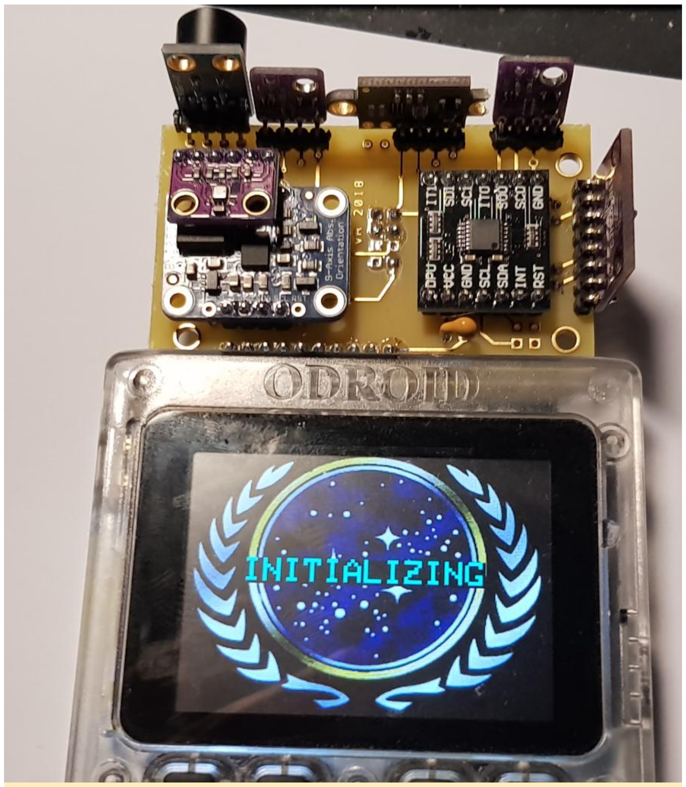


Figure 1 – Tricorder

For those of you who do not know what a Tricorder is, allow me to explain: In the newer Star Trek series, characters often carry a mobile device used for things like measuring rips in the space-time continuum and declaring “He’s dead, Jim.”

Figure 2 – Star Trek Tricorder

Although I earn my money by dealing in software, I have always had an affinity for hardware: wiring chips and other digital components together. I began with circuits based on standard TTL and CMOS logic chips.

One day, I discovered the ATMEL AVR programmable series. My journey continued, and I discovered the coolness of Espressif's ESP8266. After awhile, I found the ESP32 (WROOM32) for my projects.

I had already done projects involving sensors (i.e. BME280) using the I2C bus. I also had projects that used displays and buttons for visualization and control. What I never had was a good combination of those things, packaged together in a nice case with a battery.

Then Hardkernel popped up with the ODROID-GO– the perfect combination for me. I've had an idea for a multisensory device for quite a while. Now was the time to embark on such a project.

Basic Concept

The ODROID-GO has an expansion header with 10 Pins. Three of them are power, one is not connected and some are for SPI (which would interfere with the performance of the display). At least two of the pins– GIO15 and GPIO4–were not used by anything else. Just enough for I2C. With the ESP32, I2C can be mapped to almost any IO pin. There is no static mapping. I2C was it!

Now I had to figure out which sensors work with I2C. There are lots of them. After some googling I ended up choosing these:

- BNO055 for orientation (roll, pitch, yaw) and acceleration.
- BME280 for ambient pressure, temperature, and humidity.
- VL53L0X for distance measurement (0-120cm).
- VEML6040 for LIGHT measurement (RGB, LUX).
- VEML6075 for UV measurement (UVA, UVB => UV Index).
- CCS811 to measure CO2 and VOC gas concentrations.
- Mics6417 to measure eight more gases and their concentration.
- MLX90416 to measure the temperature (IR) of objects -70 to 300 degrees Celsius (like those contactless thermometers).

Buying

Where to buy the sensors? I decided to look to eBay and its countless Chinese sellers. It takes about a month for the sensors to arrive, but they tend to have the cheapest offers. Of course, I ordered multiples of each type in case I burned one of them. I wouldn't want to wait another month to get replacements. One usually pays about 3-6€ for a sensor. The exception were the Mics, at about 50€.

Prototyping

After the sensors arrived from China I checked out each of them by attaching them to the ODROID-GO via breadboard, writing little test program to produce serial output and display values on the ODROID's screen.

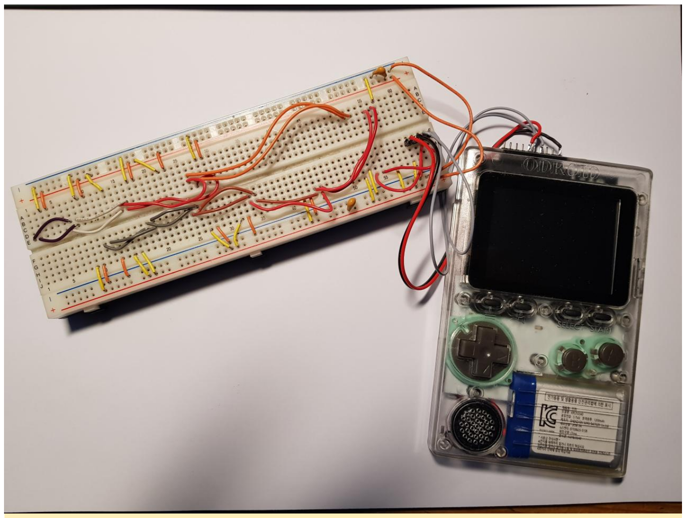


Figure 3 – Breadboard

I also created some experimental soldered boards to get a better feeling of how it would look in the end.

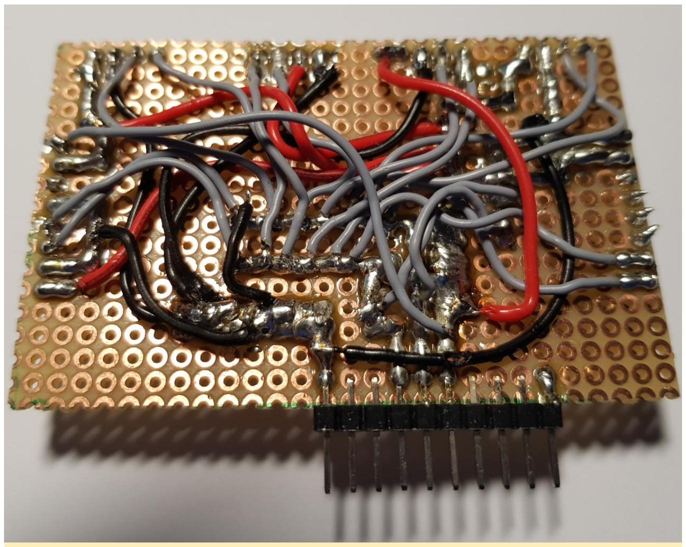


Figure 4 – Experimental_1

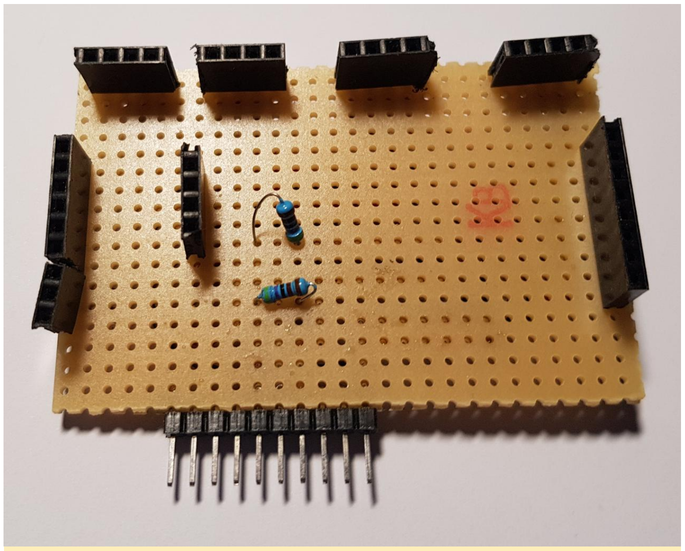


Figure 5 – Experimental_2

Schematics (Overall Wiring)

In general, wiring I2C sensors is pretty simple. I2C is a bus type interface. Just connect all SCL and SCK pins together and wire these 2 connections to VCC via resistors (Pullup).

Additionally, provide power (VCC, GND). The standard voltage for sensors is 3.3V. Luckily, the ODROID-GO provides 5V AND 3.3V. Each device in a I2C bus has its own ID. That way the controller (master) can address each sensor (slaves) on the bus. Besides the four pins mentioned, some sensors have additional pins that influence their behavior or let them respond to different I2C addresses.

In my case, it was not that simple to connect all sensors since the VEML sensors use the same I2C address of 0x10. Instead of using tricky logic to get around it, I decided to use an I2C Switch (TCA9543a). The one I chose has three I2C ports. One port communicates with the ESP32 and is the "input" port, while the other two are connected to the sensors. The VEMLs are placed at different ports. The switch can be programmed to hand through the I2C communication to either port 1 or 2.

From a programming perspective, you have to tell the switch to activate port 1 or port 2 and then query the sensors connected to that port. Switch over to the other port and query the rest of the sensors. Not really a big deal.

In the end I came up with this schematic:

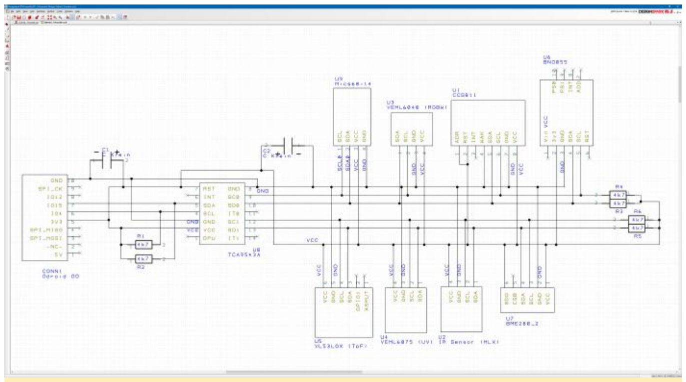


Figure 6 – Schematics

I used DesignSparkPCB for all my schematics and then later for transforming the schematics into a PCB design. It is free and I highly recommend it. You will be required to register, but this seems to be mandatory for most things these days and for the value you get out of it, registration is worth it.

One important factor for PCB software is the ability to teach it your own components. I tried to find component libraries for the sensors I used, but couldn't find any that were freely available. I gave up and just designed them myself, which is possible with this program.

Printed Circuit Board (PCB)

From the schematics you can create a PCB in DesignSparkPCB. Due to an excess of wiring, I don't think a single layer PCB would be possible. It would seem that a two-sided PCB would be necessary.

Many PCB tools offer automatic wiring. Very often this produces strange wirings, so I opted to route the wires myself. DesignsparkPCB, like all the other tools, helps keep you from forgetting wires.

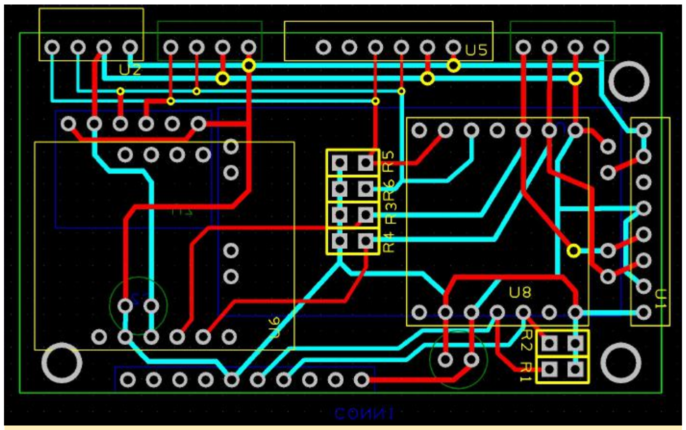


Figure 7 – PCB Layout

Red and cyan indicate the wiring on the top and bottom layer. Yellow and blue is for documentation purposes and can be printed on the PCB. That way you easily see which device will be placed and soldered in which position.

Having completed the PCB layout, it now needs to be produced. Some people do this on their own. In the past I also did it on my own, but I never liked the chemistry behind it. Additionally, there are 2 layers involved, which requires a very precise alignment of the layers. I doubt that I would be able to handle this. I decided to have it done by a professional company for about 25€.

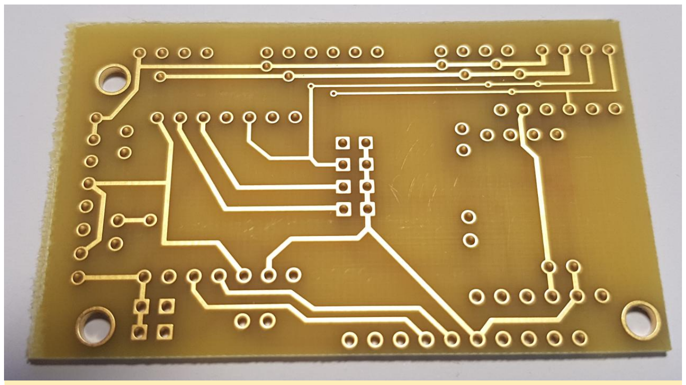


Figure 8 – PCB 1

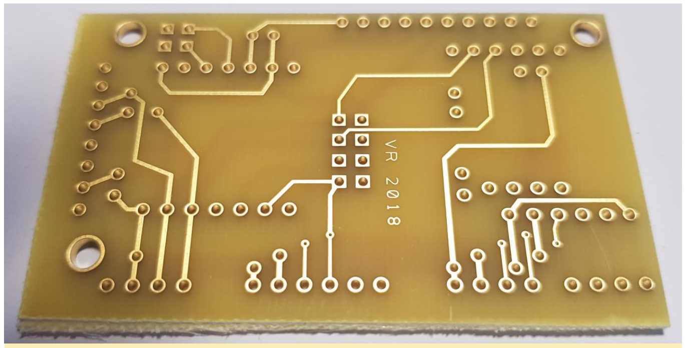


Figure 9 – PCB 2

To lower the costs for the production I decided against solder resist and documentation printing.

Mounting time

Finally having the PCB in my hand, the mounting of the sensors via soldering was done easily.

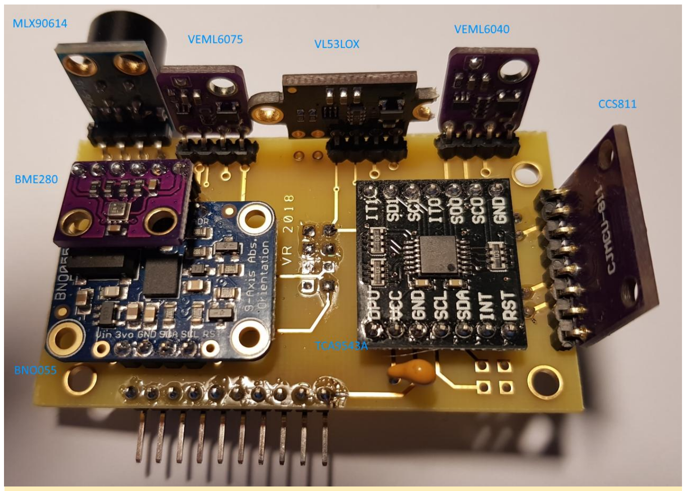


Figure 10 – Assembly Soldering

Doing my first software tests to see if all sensors could be found by the ESP32, I noticed that I made a mistake in the layout. I intended to have the BME280 on the bottom side of the PCB. Having it soldered on the wrong side, VCC and GND were connected to the wrong pins. The BME quit in smoke. Shit happens, 4€ gone. That is why you order more than one. Luckily, this mistake did not affect the other sensors or the ODROID-GO. The Mics gas sensor costs about 50€ and I only have one of them. Burning this one would hurt way more. The issue was easily resolved. The next BME280 I just soldered on top of the BNO055 orientation sensor.

Software (the UI design)

I am not a designer. Making things pretty is not my thing. I usually work on a functional level. In this project, I wanted to prove myself wrong and do a nice user interface for the Tricorder.

From a functional point of view, it was pretty clear that it is not technically feasible to do a full repaint of a whole screen (320×240 pixel) just to refresh some measurement values, maybe several times a second. The display is just not fast enough for that. Heavy flickering would be the result. Nevertheless, some cool graphic elements would be nice in the UI. This led to the design having some full-size screens with JPG background images and empty graphical areas serving as placeholders inside. In a case where the screen is rendered for the first time, the full background image is drawn and then the sensor values are pasted in. For all subsequent value refreshes, only the placeholder areas for the values have to be redrawn. This improves the responsiveness of the UI.

It was also obvious that there were too many measurement values to put them all into one screen at the same time. I introduced screens that could be rotated using the "A" and "B" button.

The hardest thing was to come up with a particular style for the screen images. I made several attempts and threw them all away. Then an idea popped up. It is a Star Trek Tricorder, so I would do it the Star Trek way. I searched Google Images for Star Trek Tricorder and tons of inspiring pictures appeared. From then on, the design style was clear.

I tried several freely available drawing programs, but ended up with paint.net, which I use for most of my graphical needs.

Software (the control logic)

You can program the ESP32 using Espressif's SDK (ESP-IDF) directly in C++, but the provided functionality seemed to be on a very basic level. Arduino is a very popular ecosystem to provide some abstraction from the low level while still being C/C++. Another advantage of using Arduino is the fact that a big bunch of libraries exist for all sensors making it very easy to use them.

For Arduino you can use the Arduino IDE, which I started with a few years ago. Then I stumbled about PlatformIO. It comes as plugin for the ATOM or the VS-CODE editor. I used both and found that I liked VS-CODE more.

Getting into detail about how to code for ESP32 in the Arduino world with VS-CODE would bloat this article. ODROID magazine already had articles about coding for the ODROID-GO. It may be more interesting to mention the things you usually do not do in an implementation for the ESP32.

I never had to deal with binary files in my code for the ESP32. The way to do it for the ODROID-GO (and ESP32 in general) is SPIFF. SPIFF is a file system for the ESP32. You can upload binary files into a special area of the ESP32's flash memory (the SPIFF partition).

For the ODROID-GO there is a collection of libraries you can use to address the hardware of the GO (speaker, buttons, display, even potential sensors). The display library supports the display of JPG files stored in the SPIFF partition.

What you have to do is to upload the JPGs into the ESP using PlatformIO. After doing that, you can program the ESP and access the uploaded files in your code by providing the filenames. That makes it very easy to fill the GO's screen with a JPG file– it needs one line of code.

Another thing I had to do was modify some sensor libraries. Those libraries are often written to work with more than one controller, thanks to the Arduino abstraction. This leads to some issues if the ESP32 works differently in some areas. The changes were not too many.

The current state of the software

The following images show the various screens I already designed. As I mentioned before, the screens can be switched back and forth using the A or B button.

Initialization Phase

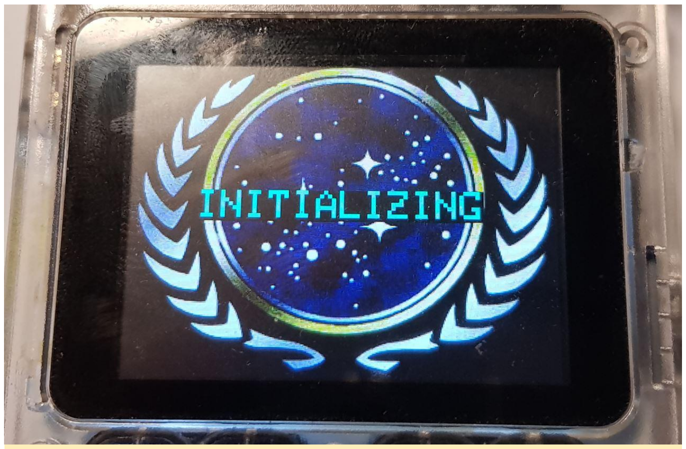


Figure 11 – Init Screen

The sensors need to be initialized. A good time for a welcome screen. I probably should have cleaned the screen before taking the photo.

Ambient Sensors

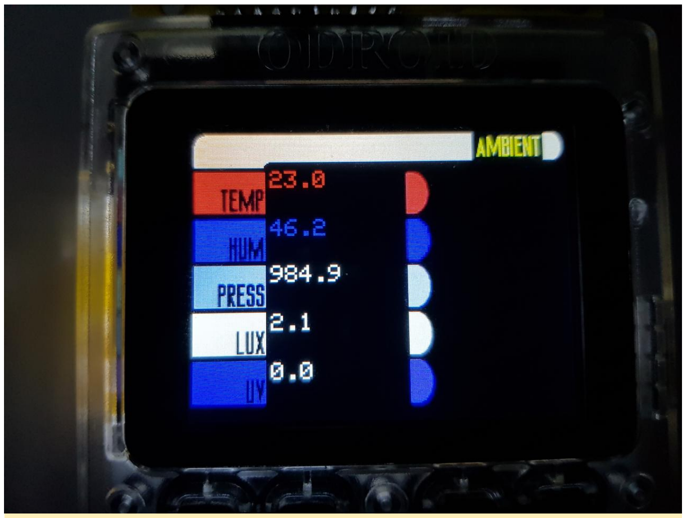


Figure 12 – Ambient Screen

The Ambient Screen holds sensor values from the BME, the VEML6040 and VEML6075. This is ambient temperature (in celsius), humidity (in %), pressure (in hPa), light intensity (in LUX) and the UV index (it's a number indicating how dangerous the current UV level is)

Light Sensor (RGB)

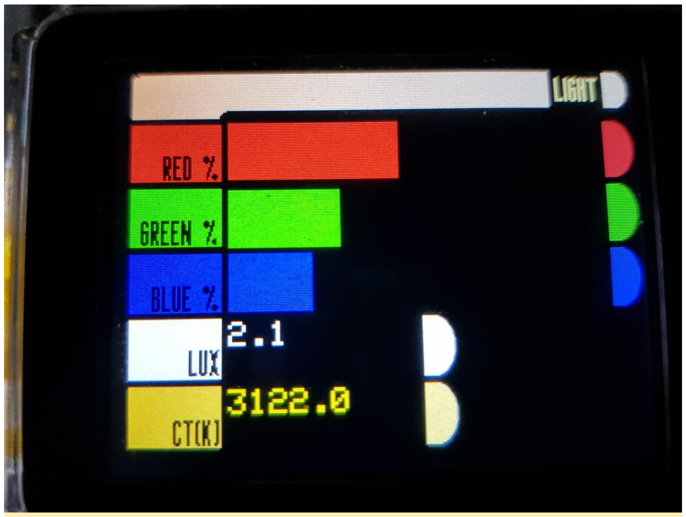


Figure 13 – Light Screen

The light view gives detailed information about the distribution of red, green, and blue in the visible light. The exact wavelengths measured by the sensor can be checked in the VEMLs datasheet.

The distribution is presented in a bar graph for red, green, and blue.

The light intensity in LUX was already present in the ambient screen and can be seen here again.

The color temperature indicates if the light is "warm" or "cold". Higher values indicate cold light (blue), lower ones indicate warm light (red)

The Gases View

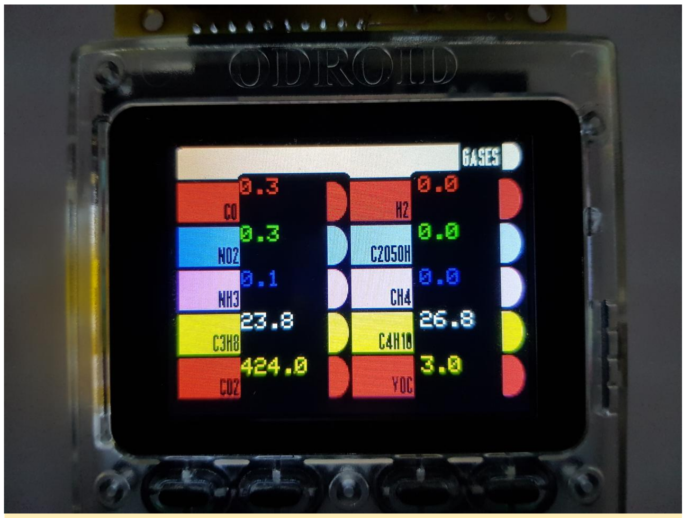


Figure 14 – Gases Screen

The gases view presents measurements from the CCS811 and the Mics sensor. Ten gases in total.

The unit of the measurements is parts per million (ppm). The gases are given by their chemical formulas since the names did not fit into the screen.

- CO – Carbon Monoxide: Can easily cause suffocation. Hard to recover from exposure.
- H2 – Hydrogen: Together with oxygen you have a high chance to blow up your house. Also used for rocket thrusters.
- NO2 – Nitrogen Dioxide: Toxic.
- C2O5OH – Ethanol: Alcohol. I like that stuff in various forms.
- NH3 – Ammoniac: Intense smell. One source can be poop. Not healthy of course.
- CH4 – Methane: Worse than CO2 in terms of greenhouse gas. Inflammable if O2 is present.
- C3H8 – Propane: People use it to fire things.
- C4H10 – Butane: Also inflammable. Camping gas.
- CO2 – Carbon Dioxide: That's what we exhale and what plants need to live. Causes trouble as a greenhouse gas.
- VOC – Volatile Compound Gases: Substances in gaseous form at room temperature. It is not precisely defined which gases are detected. It is just an indicator. The higher the value the worse it is.

In the screenshot you see that there seems to be Propane and Butane around the Tricorder. This is due to the fact that the Mics gas sensor needs some heat up time before delivering accurate values. I didn't want to wait 10 minutes to take the picture.

Distance

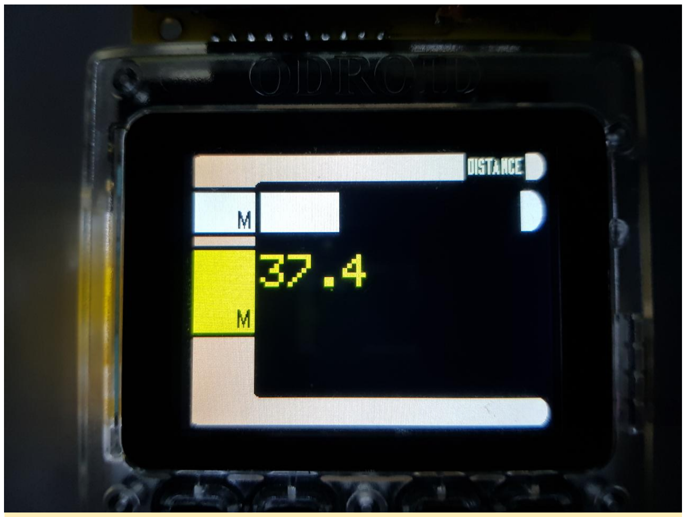


Figure 15 – Distance Screen

The distance between the Tricorder and an object it is pointed at can range 0 to 120cm. It is displayed in form of a number a bar graph (the white bar).

Temperature

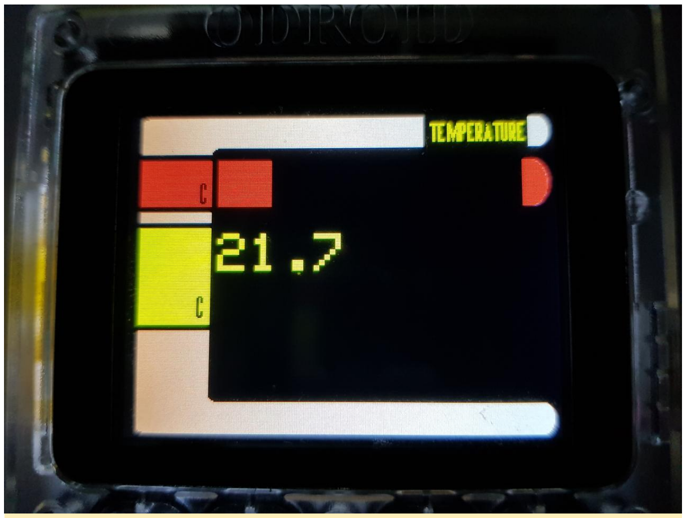


Figure 16 – Temperature Screen

The temperature of an object is displayed the same way as distance. It is given as a number and a bar graph (red bar). Unit is Celsius.

Orientation

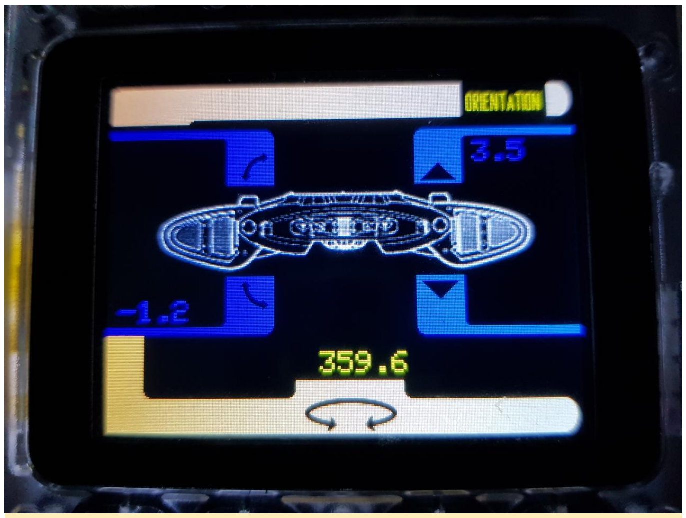


Figure 17 – Orientation Screen

The orientation was the hardest thing to design a page for. Simple numbers do not provide an intuitive meaning. The current solution is surely not the best. I hope to come up with some better ideas.

To the left, the roll is displayed. If the Tricorder is rolled counterclockwise, then the blue number to the left is displayed at the bottom position (as in the screenshot). If rolled clockwise the number would be displayed in the upper position, as the angle in degrees (0-180).

For the pitch, it is the same. If the Tricorder is directed upwards, the number indicating the rotational angle is displayed in the top position (as in the screenshot). If directed downwards, the number is displayed in the lower position.

The yaw (yellow) is just the angle the Tricorder points to (north, south, east, west), from 0-359.9 degrees. This can be used as compass.

Calibration

The sensors deliver measurement values. How accurate are those measurements? Without a reference you can't tell. It is easy to check distance measurement. Temperature becomes harder. The light intensity also requires more effort–for instance, finding a referential measurement device. Almost impossible to verify are the gas concentrations. I intend to see how I can improve here.

Implement things like hold/min/max

If you know multimeters, those devices often have the ability to track minimum, maximum or average values. Freezing the display in order to display a measurement for a longer time is also very common. The Tricorder could need that too. This will make more use of the ODROID's buttons.

Wireless measurement export?

Measurements are currently only displayed on the ODROID's display. The ESP32 can work with WiFi and Bluetooth (BLE and classic SPP). The Tricorder could be extended to provide the measurements "over the air". MQTT and BLE are candidates here. I did both already with the ESP32.

I currently do not see the use case for it in this project. One possible project is a "measurement box". Its measurements could be displayed using your mobile device. However, for this, the ODROID-GO would not be needed.

Acceleration

I currently only use the BNO055 sensor for roll, pitch, and yaw. It is capable of doing more, such as acceleration measurements (g-forces).

Case

I own a 3D printer. I surely have the plan to create a case for the Tricorder's electronics that somehow snaps to the ODROID-GO case.

Building a Commodore 64 Emulator

December 1, 2018 By AreaScout Gaming, ODROID-C2

This emulator allows one play to games designed for the Commodore 64 8-bit system. It has been ported to the ODROID-C2 Single Board Computer (SBC).

Figure 01 – Commodore 64 Emulator

Following are the steps involved in obtaining the source code, applying relevant patches and running the emulator. I was able to contribute some patches to the master code base. It was not easy to solve the various issues, but with some good help from the project maintainers, it is now available for all.

Checkout the source and apply a patch

First, we need to obtain some prerequisites:

```
$ sudo apt-get install bison
```

Then checkout the source and apply the no-border patch for VIC-II Commodore machines if you wish. This will remove the border of C64 and C128 machine models – the games are way better to view without it. This is more of a quick and dirty way to do it, a better approach would be to add it to the libretro config. If a game draws inside those borders it will not work and the application will crash. However, a lot of games do not draw borders.

```
$ git clone https://github.com/libretro/vice-
libretro.git
$ cd vice-libretro
$ wget -O noborder.patch
https://pastebin.com/raw/VwtSDj50
$ patch -p1 < noborder.patch
```

You can then start to build a Commodore machine of your choice. The valid machine types include the following:

- x128
- x64
- x64sc
- x64dtv
- x64scpu
- xplus4
- xvic
- xcbm5x0
- xcbm2
- xpet

You will need to specify the EMUTYPE variable followed by the machine type, reflecting your build choice. If left unspecified, x64 (C64) is selected as the default.

```
$ make EMUTYPE=x64 -f Makefile.libretro -j7
```

If you want to build more then one machine type, do not forget to run clean (make EMUTYPE=x64 -f Makefile.libretro -j7 clean) on the project otherwise the core will not work.

Apply RetroArch config

To apply the RetroArch configuration, copy the binary into RetroArch core folder:

```
$ cp vice_x64_libretro.so
~/.config/retroarch/cores/ .
```

Start RetroArch select the vice core, then either start the core with or without a game. Press the Guide button on your game controller or F1 on the keyboard and scroll down to Options, select it and disable DriveTrueEmulation->OFF, and set Controller0Type to “joystick”

I also enable a Aspect Ratio of 16:10, which I think is a good compromise between 4:3 and 16:9:

```
Settings -> Video -> Aspect Ratio -> 16:10
```

With the Start button, you activate the nuklear GUI settings (select button has to pressed once to activate mouse). From there, you can choose the C64 Joyport, machine cpu, sid type and more. The Onscreen keyboard is activated with the “X” button (Xbox layout).

References

http://vice-emu.sourceforge.net/
https://forum.odroid.com/viewtopic.php?f=98&t=32173#p233998
https://youtu.be/ItkppnXWd9U

Coding Camp – Part 10: Measure the distance with Ultrasonic

December 1, 2018 By Justin Lee ODROID-C2, Tinkering, Tutorial

Let us learn how to use GPIO output, IRQ input and system timer with a Ultrasonic distance measuring module. Note that the distance sensor is additionally required (https://wiki.odroid.com/odroid_go/arduino/31_ultrasonic_distance_meter).

Figure 1 – You can have a portable Ultrasonic distance meter in your hand

Requirements

Make sure that you have these products:

- ODROID-GO
- Ultrasonic Ranging Module HC – SR04
- Jumper Wires and a Breadboard full size or half size
- An auxiliary battery if using it portable
- This module requires 5V input power, but ODROID-GO outputs 3V3 for the power pin(#6) on its header pins. Thus, it is required to use the 5V USB VBUS pin(#10) with an external USB power source.
- Alternatively, you can use a Step-up DC-DC converter and a level shifter.
- A MicroUSB cable

You will also need to set up the development environment for Arduino on your system.

Hardware Setup

Please refer to Figure 2 when you set up your hardware. We used the following parts:

- Ultrasonic sensor: HC-SR04

- Step-up DC-DC 3V3 to 12V(set to 5V for now) converter: MT3608
- Logic level converter: BSS138
- ODROID-GO: fritzing_odroid-go.fzpz
- Mini-360 DC-DC Buck converter: mini-360_dc-dc_buck_converter.fzpz
- HC-SR04: hc-sr04.fzpz

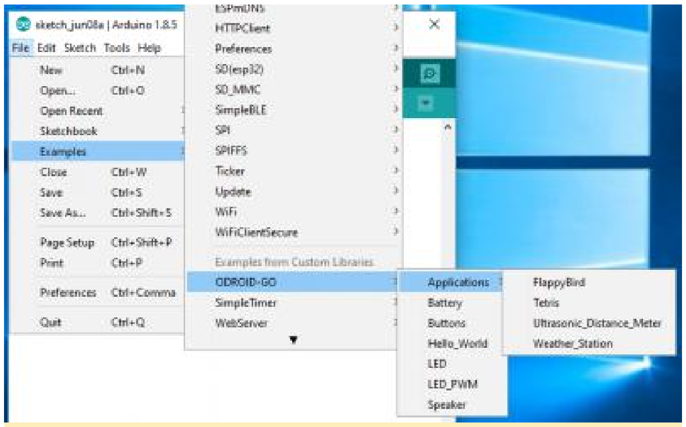


Figure 2 – Wire Diagram

Next, download the Fritzing example file from odroid-go-ultrasonic-sensor.fzz.

Import and Compile, Upload to ODROID-GO

Click the Files → Examples → ODROID-GO → Applications → Ultrasonic_Distance_Meter menu to import and press CTRL-U to compile/upload.

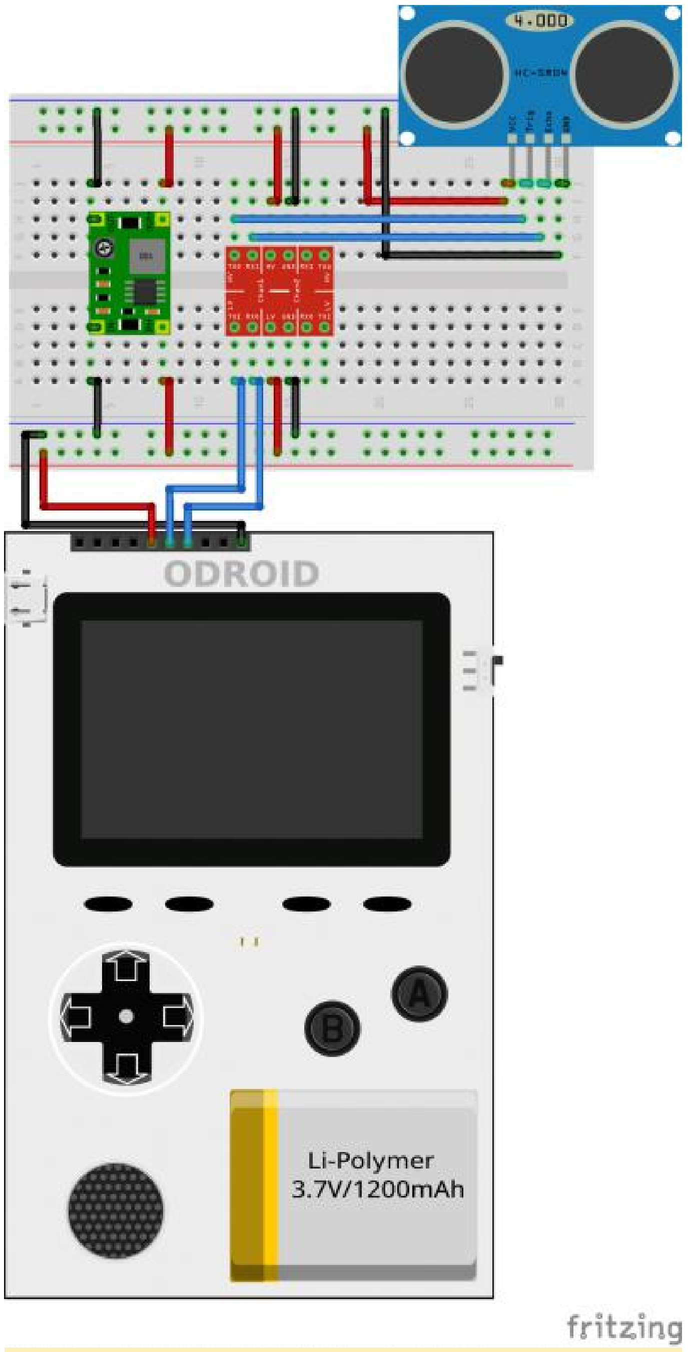


Figure 3 – Load Arduino Ultrasonic Distance Meter Application

Uploading is complete when the message “Hard resetting via RTS pin…” is seen.

Testing

After the upload completes, ODROID-GO reboots automatically. The screen shows a measured distance in inch, cm units when an obstacle is detected at the front of the Ultrasonic sensor. If the measurement conditions are not met, such as a distance that is too far or too short, the result text on the screen will be colored red. If it measures normally, the text will be colored green.

Introducing NEMS Linux: Part 3 – Configuring Service Monitors on NEMS Linux

December 1, 2018 By Robbie Ferguson Linux, Tutorial

This is the third part in an introductory series to NEMS Linux: the Nagios Enterprise Monitoring Server for ODROID devices. If you haven't read the first two parts (October and November issues of ODROID Magazine), please start there as the lessons build upon each other. My intention with these articles has been to introduce you to NEMS Linux in such a way as to arm you with useful knowledge that gets you up and running immediately. These aren't intended to appear as documentation, but rather a technical article that gives you ideas as to how NEMS Linux can be used in your environment. This month, however, we'll be geeking out together as I provide two key exercises that you may find useful in monitoring your network assets with NEMS Linux.

In this month's exercises, you'll learn the skills needed to configure NEMS Linux to perform the following:

- Tell if your web site is up, and notify you if it has been down for more than 10 minutes: monitor your own, your customers' or any http/https web site for uptime or slow response time.
- Monitor the state of a specific TCP/UDP port on a network connected device, and notify you if it stops responding: tell if your local blockchain node has stopped responding on port 8333, Apache2 has stopped responding on port 443, or monitor the state of openssh running on your server on port 22. These are just examples. The options are limitless.

Understanding Notification Definitions

Before we get into our exercises, a quick glossary will help you understand what the single-character notification options mean. Refer back to this list during the exercises to understand what we're actually doing when we specify, for example, **w,u,c,r,f**.

When you see **w,u,c,r,f,n**, these are the definitions:

- w Notify if in warning state,
- u Notify if in unknown state,
- c Notify if in critical state,
- r Notify if recovered from a previously bad state,
- f Notify if the service is flapping (on and off and on and off)
- n Never notify.

When you see **d,u,r,f,s,n**, these are the definitions:

- d Notify if host is down,
- u Notify if host is unreachable (eg. Internet down),
- r Notify upon recovery,
- f Notify if the host is flapping,
- s Notify if a scheduled service downtime begins or ends,
- n Never notify.

Exercise 1: Monitor Your Web Site with check_http

Your web site is the face of your business. If it ever goes down for any reason, or becomes sluggish, it's important to be proactive in remedying the situation. The only thing worse than having a customer contact you to let you know your web site is down is realizing it might have been down for a week and the customers during that time didn't let you know. They just went elsewhere. Having your web site become sluggish or unresponsive can also damage your organic SEO standings.

NEMS Linux can keep a close eye on your web site and send you an alert by email, Telegram or Pushover if your site goes offline, or becomes unresponsive or sluggish. This also makes NEMS Linux a fantastic tool for web designers and hosts who want to ensure their customer sites are always up so the customer doesn't notice any downtime. If your site is hosted over SSL, NEMS can even notify you if your certificate has expired – or is about to expire. There are just so many options since NEMS Linux has been built to monitor everything.

In our first exercise this month, we'll use the built-in check_http command. For my example, I'll use https://nemslinux.com/ – I would suggest you do the same for the sake of the lesson, and then try changing the Host to your own domain once you understand how everything is connected. Remember, I'm expecting you've already completed the previous two articles in ODROID Magazine's October and November 2018 issues. If not, go back and read those first. If you're ready, let's get right into it! It may appear onerous as you glance over the following 6 steps, but keep in mind once you create your config, you can reuse it for as many web site hosts as you like by simply assigning your host to the web_site_ssl hostgroup, which you'll learn to create below.

Open NEMS NConf and follow these steps:

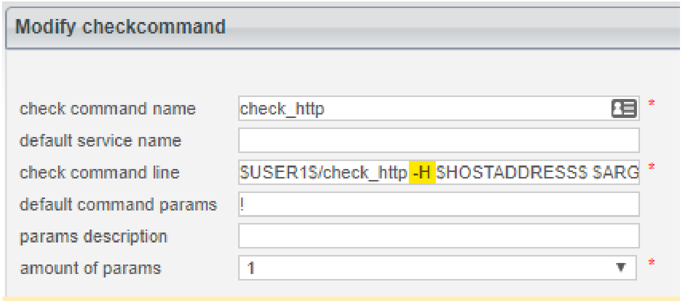
1. In preparation, we need to make sure our check command is ready for our use case. While the default is changing in NEMS 1.5, if you're on NEMS 1.4.1 you will need to change the check command to use hostname checks rather than IP address checks.

a. Show your checkcommands list.

b. Edit check_http

c. Currently the command line uses -I %HOSTNAME%, with -I meaning "IP Address". Change that to -H (hostname) so it now reads … -H %HOSTNAME% … Now we can use our web site's hostname or an IP address for the check_http command.

d. Save your change.



Modify checkcommand

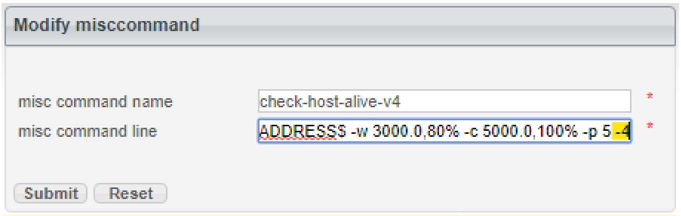
check command name	check_http
default service name	
check command line	$USER1$/check_http -H $HOSTADDRESS$ $ARG
default command params	!
params description	
amount of params	1

Figure 1 – Modify check_http to use Hostname Instead of IP Address

2. Next up, we need to setup our check-host-alive command, which is used to ping hosts to determine if they are up or down. My web site hostname will only respond on IPv4, though the default check-host-alive command in NEMS 1.4.1 uses IPv6. Rather than editing the sample command, let's add a new one based upon it, but this one will only use IPv4. That

way, we can still use the old command if we need IPv6 for a different device.

a. Show the misccommands list.
b. Edit check-host-alive
c. Highlight and copy the entire command line to your clipboard.
d. Click Add next to misccommands to add a new command.
e. Name your new command check-host-alive-ipv4
f. Paste the command line from your clipboard.
g. At the very end of the command line, simply add a space, followed by -4 to tell it to use IPv4 for this check.
h. Save the new command.

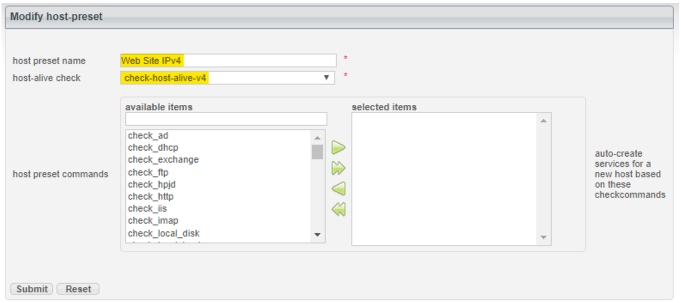

Modify misccommand

misc command name	check-host-alive-v4
misc command line	ADDRESS$ -w 3000.0,80% -c 5000.0,100% -p 5 -4

Figure 2 – Create New misccommand to check-host-alive Using IPv4

3. Our commands are ready for us, so now it's time to setup our hostpreset. We want to create one for IPv4 Web Sites. That way, we can reuse the preset for every web site we want to monitor with NEMS Linux.

a. Add a new host preset.
b. Name your preset Web Site IPv4
c. Set the host alive check to the new command you created in Step 2: check-host-alive-ipv4
d. Save your host preset.

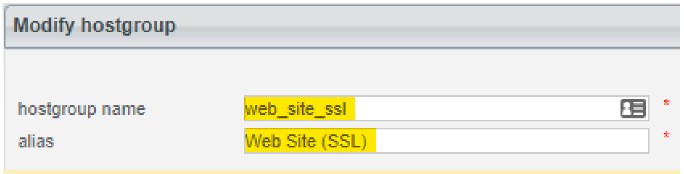

Modify host-preset

host preset name	Web Site IPv4
host-alive check	check-host-alive-v4

Figure 3 – New Host Preset for IPv4 Web Sites

4. So far, everything we've done can be reused for any web site whose hostname resolves to an IPv4 address. From here forward however, we'll be setting up our host group specifically for a secure (SSL) web site.

a. Add a new hostgroup.
b. Call this web_site_ssl
c. Leave everything else as is and save your new hostgroup.


Modify hostgroup

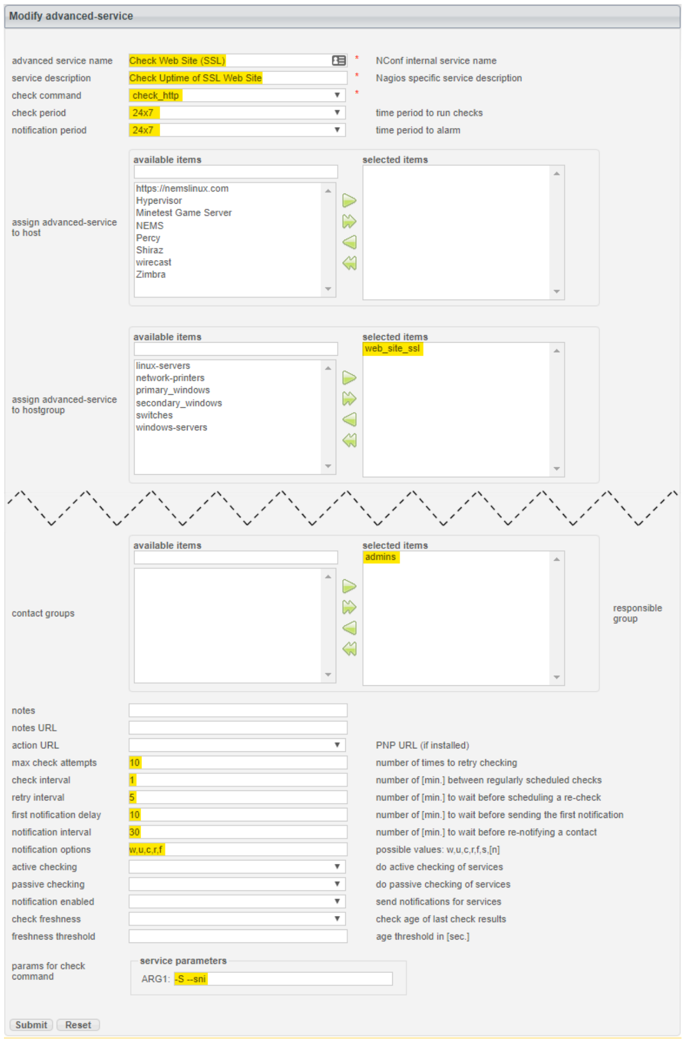
hostgroup name	web_site_ssl
alias	Web Site (SSL)

Figure 4 – New hostgroup for web_site_ssl

5. Why would we create a new hostgroup if it has no settings beyond a name? Well, this is where the magic happens. We now have a check command, a check host alive command, a host preset and a hostgroup. Now, we can link them all together, starting with an Advanced Service. Remember, the idea here is that everything we do can be assigned to as many hosts as we like. No having to redo all this for the next web site.

a. Click Add next to Advanced Services.
b. Name your service: Web Site (SSL)
c. Give it an alias: Uptime of SSL Web Site
d. Set the check period and notification period to 24×7
e. In assign advanced-service to hostgroup, highlight the hostgroup we created web_site_ssl and press the green arrow to add it to the selected items list.
f. Under contact groups be sure to add admins as well. Otherwise, you won't receive notifications.
g. Set your notifications as follows: max check attempts: 10 ; check interval: 1 ; retry interval: 5 ; first notification delay: 10 ; notification interval: 30 ; notification options: w,u,c,r,f
h. Finally, set your service parameters to: -S –sni
i. Save your advanced service.

Tip: The -S tells check_http that this site is using SSL, and the –sni enables SNI (Server Name Indication) since I use CloudFlare for SSL on nemslinux.com, and therefore my resolving IP address is associated with more than one domain name. For your site, if you have any trouble, try removing SNI by simply omitting –sni. For the full documentation surrounding the check_http command, visit the NEMS Linux documentation wiki page at https://docs.nemslinux.com/check_commands/check_http



Modify advanced-service

advanced service name	Check Web Site (SSL)	*	NConf internal service name
service description	Check Uptime of SSL Web Site	*	Nagios specific service description
check command	check_http	*	
check period	24x7		time period to run checks
notification period	24x7		time period to alarm

assign advanced-service to host — available items: https://nemslinux.com, Hypervisor, Minetest Game Server, NEMS, Percy, Shiraz, wirecast, Zimbra; selected items: (none)

assign advanced-service to hostgroup — available items: linux-servers, network-printers, primary_windows, secondary_windows, switches, windows-servers; selected items: web_site_ssl

contact groups — available items: (none); selected items: admins — responsible group

notes		
notes URL		
action URL		PNP URL (if installed)
max check attempts	10	number of times to retry checking
check interval	1	number of [min.] between regularly scheduled checks
retry interval	5	number of [min.] to wait before scheduling a re-check
first notification delay	10	number of [min.] to wait before sending the first notification
notification interval	30	number of [min.] to wait before re-notifying a contact
notification options	w,u,c,r,f	possible values: w,u,c,r,f,s,[n]
active checking		do active checking of services
passive checking		do passive checking of services
notification enabled		send notifications for services
check freshness		check age of last check results
freshness threshold		age threshold in [sec.]

params for check command — service parameters — ARG1: -S --sni

Figure 5 – Creating an Advanced Service to Check SSL Web Sites

6. Finally, let's add our web site host. From now on, this is the only step you have to take to add more sites to your NEMS Linux server.

a. Add a new host.

b. Set the following:

c. hostname: https://nemslinux.com

d. alias: NEMS Web Site

e. address: nemslinux.com

f. OS: Linux

g. host preset: Web Site IPv4 (See what we did there?)

h. monitored by: Default Nagios

i. host is collector: no

j. check period: 24×7

k. notification period: 24×7

l. max check attempts: 10

m. check interval: 1

n. retry interval: 5

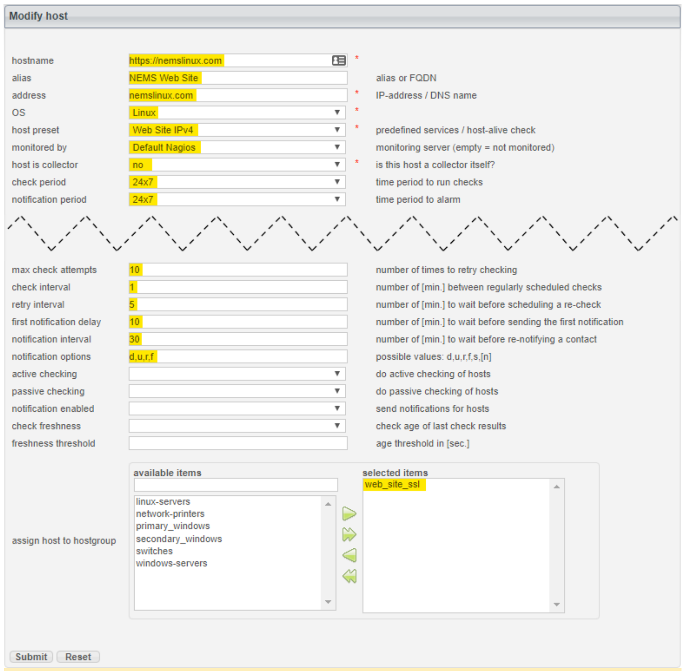
o. first notification delay: 10

p. notification interval: 30

q. notification options: d,u,r,f

r. assign host to hostgroup (are you ready for this?): web_site_ssl

s. Save.



Modify host

hostname	https://nemslinux.com	*	
alias	NEMS Web Site		alias or FQDN
address	nemslinux.com	*	IP-address / DNS name
OS	Linux	*	
host preset	Web Site IPv4	*	predefined services / host-alive check
monitored by	Default Nagios		monitoring server (empty = not monitored)
host is collector	no	*	is this host a collector itself?
check period	24x7		time period to run checks
notification period	24x7		time period to alarm

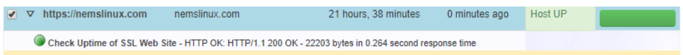
max check attempts	10	number of times to retry checking
check interval	1	number of [min.] between regularly scheduled checks
retry interval	5	number of [min.] to wait before scheduling a re-check
first notification delay	10	number of [min.] to wait before sending the first notification
notification interval	30	number of [min.] to wait before re-notifying a contact
notification options	d,u,r,f	possible values: d,u,r,f,s,[n]
active checking		do active checking of hosts
passive checking		do passive checking of hosts
notification enabled		send notifications for hosts
check freshness		check age of last check results
freshness threshold		age threshold in [sec.]

assign host to hostgroup — available items: linux-servers, network-printers, primary_windows, secondary_windows, switches, windows-servers; selected items: web_site_ssl

Figure 6 – Creating a Host to Monitor IPv4 SSL Web Site

7. Generate your config.

If you followed the steps correctly and my web site is up, Adagios should report all is well. To test what would happen if it were to start failing, change the hostname in the Host to nemslinux.com1 (which obviously will not respond), and then generate your config again. Once you feel ready, change the Host to your own web site. If your site is SSL, you should only need to change the hostname, alias and address of the host (Step 6). If it's not SSL, repeat Step 4, but this time create a new hostgroup called web_site_no_ssl, and then repeat Step 5, this time, creating a new Advanced Service called Web Site (Non-SSL), assign it (5.e) to Web Site (Non-SSL) and leave off the SSL parameters in 5.h.



```
https://nemslinux.com    nemslinux.com    21 hours, 38 minutes    0 minutes ago    Host UP
    Check Uptime of SSL Web Site - HTTP OK: HTTP/1.1 200 OK - 22203 bytes in 0.264 second response time
```

Figure 7 – NEMS Adagios Shows nemslinux.com is UP

Exercise 2: Monitor A Non-Standard Port with check_tcp

Here's a bonus exercise which will help you monitor the uptime of any TCP/UDP port. NEMS Linux includes a dummy port listener running on port 9590. The port listener is cleverly called 9590, and does nothing other than reply that it is up. This can be used to simulate a port on another device. Let's setup a service monitor on the NEMS host to warn us if port 9590 ever goes offline.

- On the left menu of NConf, you'll see "Services". Click "Add".
- Set the Service Name to: 9590
- Leave Service Enabled set to: Yes
- Set the Check Command to: check_tcp
- Set Assigned to Host to: NEMS (this host comes pre-installed)
- Leave Check Period set to: 24×7
- Set Notification Period to: 24×7
- Leave Service Templates as is, none selected.
- Under Contact Groups highlight the 'admins' group and press the arrow pointed right to move it to Selected Items.
- Leave Notes, Notes URL and Action URL blank.
- Set Max Check Attempts to: 30
- Set Check Interval to: 1
- Set Retry Interval to: 1
- Set First Notification Delay to: 5
- Set Notification Interval to: 15
- Set Notification Options to: w,u,c,r,f,s
- Leave Active Checking, Passive Checking, Notification Enabled, Check Freshness and Freshness Threshold blank.
- Leave Assign Service to servicegroup as is, none selected.
- Set Params for check command to the port number: 9590
- Press Submit
- Press Generate Nagios Config, followed by pressing the Generate button on the next screen to deploy and activate your new configuration.

Once the new config is running, try failing the service by opening Monit Service Manager under System on the NEMS Dashboard. Click on the Process named 9590, and then click "Stop service". You'll notice in around 1 minute the status of 9590 will show as a problem in all status views (Eg., NEMS TV Dashboard, NEMS Adagios, Nagios Core), and after roughly 5 minutes you will receive a notification (assuming your notifications settings are configured). Once you have received a notification, visit NEMS Adagios to Acknowledge the outage. Then, return to Monit, open the 9590 Process, and click "Enable Monitoring". This will re-load 9590 and you'll soon see it change to a Recovered state. Once complete, try setting up a new service to monitor a real host on your network. Simply change the name of the service to something appropriate, the host in step 5 (you already know how to add new hosts if you don't already have it configured), and the port number in step 19.

Learn More

NEMS has an active Community Forum. I check in quite regularly to provide free support to users. I also offer commercial one-on-one priority support for those needing a higher level of support. NEMS Linux is free to download and use. Its source code is available on GitHub. Download NEMS Linux for ODROID at https://nemslinux.com/

You can also follow @NEMSLinux on Twitter or join us on Discord.

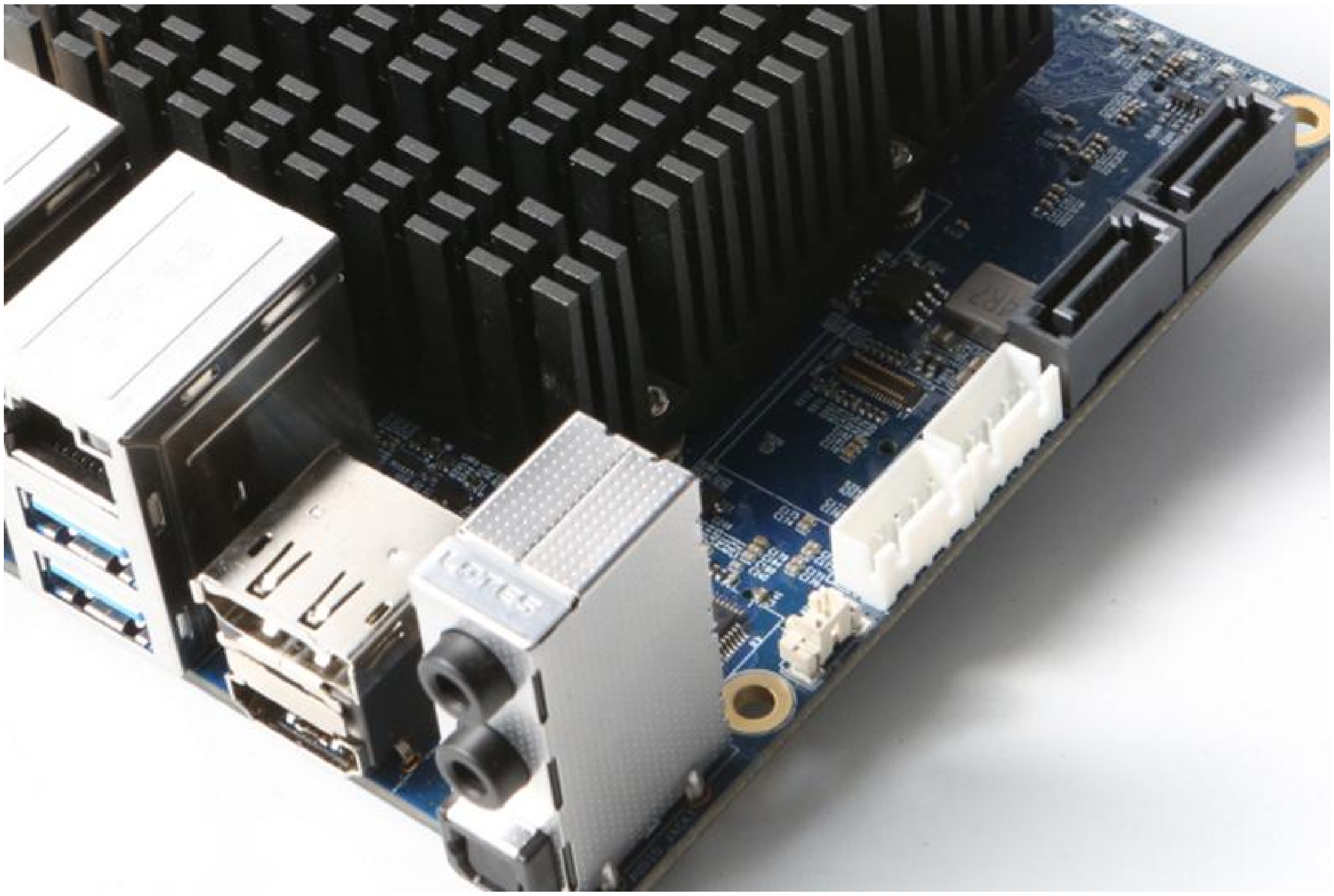
Be sure to read my article in next month's edition of ODROID Magazine as I unveil the incredible enhancements of NEMS Linux 1.5 and show you how to upgrade from NEMS 1.4.1.

About the Author

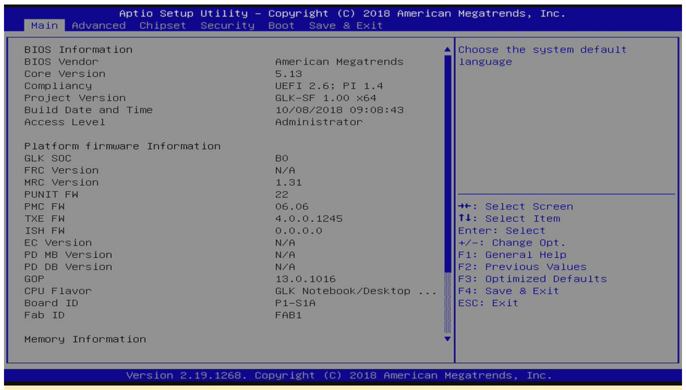
Robbie Ferguson is the host of Category5 Technology TV and author of NEMS Linux. His TV show is found at https://category5.tv/ and his blog is https://baldnerd.com/.

ODROID-H2 Part 2: Bios Features and Remote Access

December 1, 2018 By Justin Lee ODROID-H2



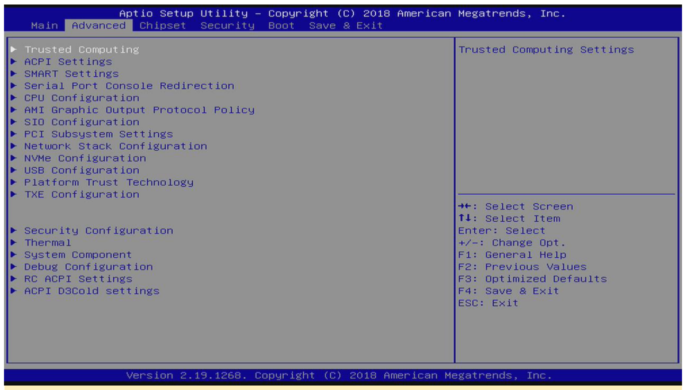
Like a generic PC, the ODROID-H2 has a soldered 8MiB BIOS Flash ROM on the board. It meets the UEFI Specification 2.6 and the PXE boot requirement. However, Intel UEFI firmware doesn't support CSM version 2.0 for legacy OS booting such as DOS, XP, Windows 7, and so on. The Main setup menu appears when you press the "Delete" key in the boot process.



```
              Aptio Setup Utility – Copyright (C) 2018 American Megatrends, Inc.
   Main  Advanced  Chipset  Security  Boot  Save & Exit

  BIOS Information                                     Choose the system default
  BIOS Vendor              American Megatrends         language
  Core Version             5.13
  Compliancy               UEFI 2.6; PI 1.4
  Project Version          GLK-SF 1.00 x64
  Build Date and Time      10/08/2018 09:08:43
  Access Level             Administrator

  Platform firmware Information
  GLK SOC                  B0
  FRC Version              N/A
  MRC Version              1.31
  PUNIT FW                 22                          →←: Select Screen
  PMC FW                   06.06                       ↑↓: Select Item
  TXE FW                   4.0.0.1245                  Enter: Select
  ISH FW                   0.0.0.0                     +/-: Change Opt.
  EC Version               N/A                         F1: General Help
  PD MB Version            N/A                         F2: Previous Values
  PD DB Version            N/A                         F3: Optimized Defaults
  GOP                      13.0.1016                   F4: Save & Exit
  CPU Flavor               GLK Notebook/Desktop ...    ESC: Exit
  Board ID                 P1-S1A
  Fab ID                   FAB1

  Memory Information

              Version 2.19.1268. Copyright (C) 2018 American Megatrends, Inc.
```

Figure 1 – Main setup menu

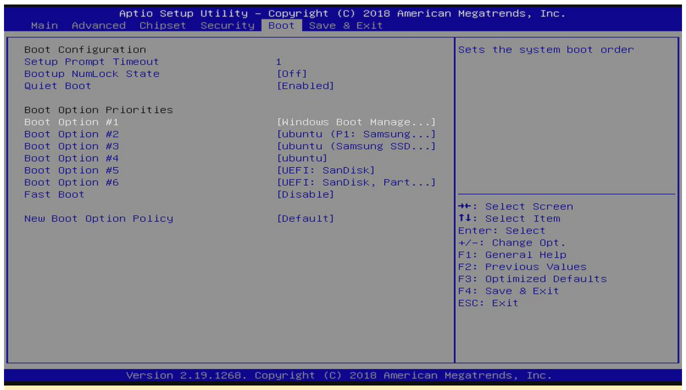
This is the Advanced setup menu:



```
              Aptio Setup Utility – Copyright (C) 2018 American Megatrends, Inc.
   Main  Advanced  Chipset  Security  Boot  Save & Exit

 ▶ Trusted Computing                                   Trusted Computing Settings
 ▶ ACPI Settings
 ▶ SMART Settings
 ▶ Serial Port Console Redirection
 ▶ CPU Configuration
 ▶ AMI Graphic Output Protocol Policy
 ▶ SIO Configuration
 ▶ PCI Subsystem Settings
 ▶ Network Stack Configuration
 ▶ NVMe Configuration
 ▶ USB Configuration
 ▶ Platform Trust Technology
 ▶ TXE Configuration
                                                       →←: Select Screen
                                                       ↑↓: Select Item
 ▶ Security Configuration                              Enter: Select
 ▶ Thermal                                             +/-: Change Opt.
 ▶ System Component                                    F1: General Help
 ▶ Debug Configuration                                 F2: Previous Values
 ▶ RC ACPI Settings                                    F3: Optimized Defaults
 ▶ ACPI D3Cold settings                                F4: Save & Exit
                                                       ESC: Exit

              Version 2.19.1268. Copyright (C) 2018 American Megatrends, Inc.
```

Figure 2 – Advanced setup menu

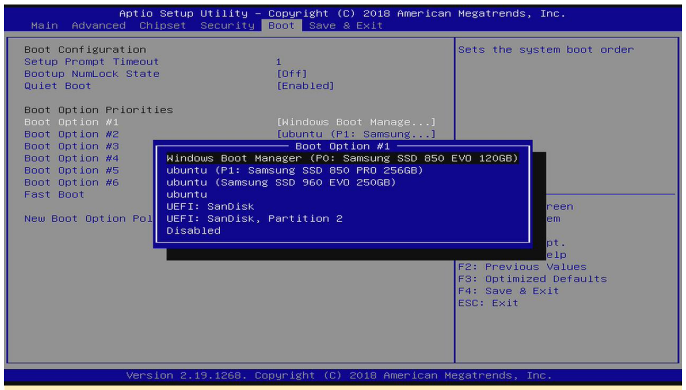
This is the Boot configuration menu that allows you to choose a boot media. ODROID-H2 can boot from eMMC, USB, SATA and NVMe storages. You can access them at the same time from the OS.



```
                 Aptio Setup Utility – Copyright (C) 2018 American Megatrends, Inc.
   Main  Advanced  Chipset  Security  Boot  Save & Exit

   Boot Configuration                                  Sets the system boot order
   Setup Prompt Timeout           1
   Bootup NumLock State           [Off]
   Quiet Boot                     [Enabled]

   Boot Option Priorities
   Boot Option #1                 [Windows Boot Manage...]
   Boot Option #2                 [ubuntu (P1: Samsung...]
   Boot Option #3                 [ubuntu (Samsung SSD...]
   Boot Option #4                 [ubuntu]
   Boot Option #5                 [UEFI: SanDisk]
   Boot Option #6                 [UEFI: SanDisk, Part...]
   Fast Boot                      [Disable]
                                                       →←: Select Screen
   New Boot Option Policy         [Default]            ↑↓: Select Item
                                                       Enter: Select
                                                       +/-: Change Opt.
                                                       F1: General Help
                                                       F2: Previous Values
                                                       F3: Optimized Defaults
                                                       F4: Save & Exit
                                                       ESC: Exit

              Version 2.19.1268. Copyright (C) 2018 American Megatrends, Inc.
```

Figure 3 – Boot configuration menu

You can change the boot priority in the Boot menu, or press F7 to temporarily choose the boot media in the boot process.



```
                 Aptio Setup Utility – Copyright (C) 2018 American Megatrends, Inc.
   Main  Advanced  Chipset  Security  Boot  Save & Exit

   Boot Configuration                                  Sets the system boot order
   Setup Prompt Timeout           1
   Bootup NumLock State           [Off]
   Quiet Boot                     [Enabled]

   Boot Option Priorities
   Boot Option #1                 [Windows Boot Manage...]
   Boot Option #2                 [ubuntu (P1: Samsung...]
   Boot Option #3     ──────── Boot Option #1 ────────
   Boot Option #4     Windows Boot Manager (P0: Samsung SSD 850 EVO 120GB)
   Boot Option #5     ubuntu (P1: Samsung SSD 850 PRO 256GB)
   Boot Option #6     ubuntu (Samsung SSD 960 EVO 250GB)
   Fast Boot          ubuntu
                      UEFI: SanDisk                          reen
   New Boot Option Pol UEFI: SanDisk, Partition 2            em
                      Disabled
                                                             pt.
                                                             elp
                                                       F2: Previous Values
                                                       F3: Optimized Defaults
                                                       F4: Save & Exit
                                                       ESC: Exit

              Version 2.19.1268. Copyright (C) 2018 American Megatrends, Inc.
```

Figure 4 – Change boot priority

Wake on LAN(WoL) feature. You can activate the WoL function on the command line. For example:

```
$ sudo ethtool -s enp3s0 wol g
```

Check the current status.

```
$ sudo ethtool enp3s0 | grep Wake
$ Supports Wake-on: pumbg
$ Wake-on: g
```

If you find g, the Wake-on-LAN feature is enabled.

If you can, wake up the ODROID-H2 board with this command from a remote PC.

```
$ powerwake 192.168.30.4
```

The WoL feature works only with an Ethernet port near the HDMI/DP port.

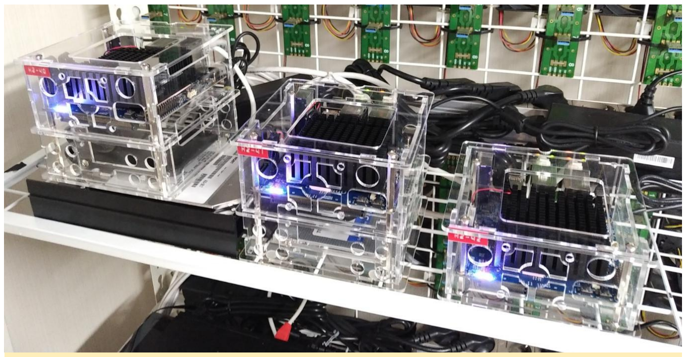
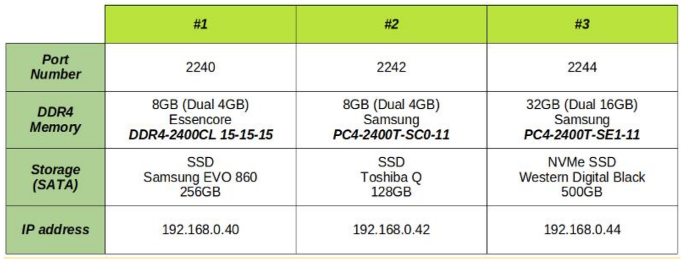


Figure 5 – Three ODROID-H2's set up with different DDR4 memory

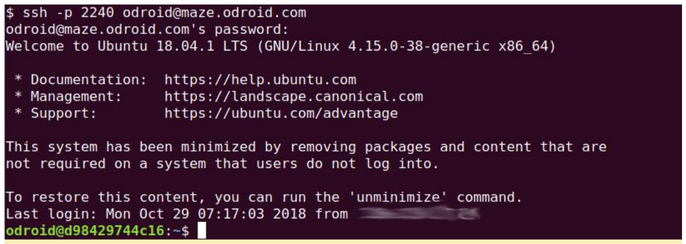
Hardkernel introduced the ODROID-BENCH in order to give users a chance to use ODROID single board computers remotely. Now we'll set up the new ODROID-H2s with a few different DDR4 memory and storage combinations.



	#1	#2	#3
Port Number	2240	2242	2244
DDR4 Memory	8GB (Dual 4GB) Essencore ***DDR4-2400CL 15-15-15***	8GB (Dual 4GB) Samsung ***PC4-2400T-SC0-11***	32GB (Dual 16GB) Samsung ***PC4-2400T-SE1-11***
Storage (SATA)	SSD Samsung EVO 860 256GB	SSD Toshiba Q 128GB	NVMe SSD Western Digital Black 500GB
IP address	192.168.0.40	192.168.0.42	192.168.0.44

Figure 6 – Configurations for each of our three ODROID-H2s

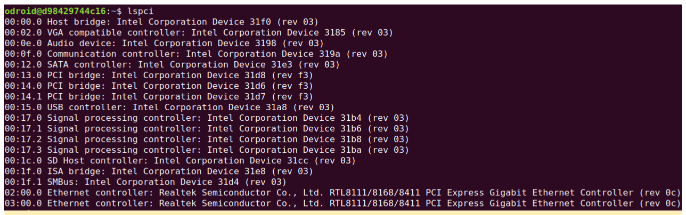
They can be accessed through "ssh" with a port number dedicated to each machine. Your access is limited to the Docker container in Ubuntu 18.04.1 and the Linux kernel 4.15.0-38-generic.



```
$ ssh -p 2240 odroid@maze.odroid.com
odroid@maze.odroid.com's password:
Welcome to Ubuntu 18.04.1 LTS (GNU/Linux 4.15.0-38-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.
Last login: Mon Oct 29 07:17:03 2018 from [redacted]
odroid@d98429744c16:~$
```

Figure 7 – Docker for Ubuntu 18.04.1

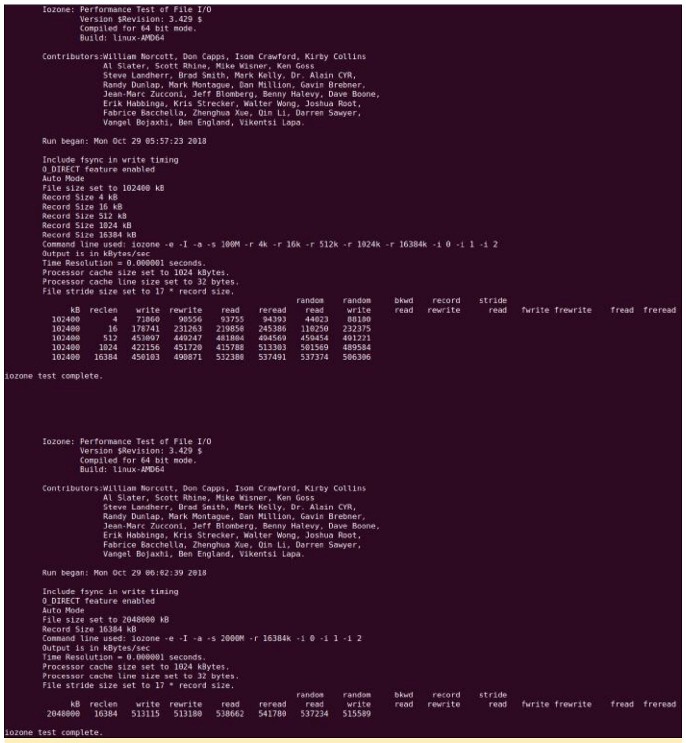
Still, you can run a bunch of system commands with root privileges and grab the hardware information. You can even run a benchmark tool, but the score will be very close or slightly lower to the one tested on the native environment since your access is in the container.



```
odroid@d98429744c16:~$ lspci
00:00.0 Host bridge: Intel Corporation Device 31f0 (rev 03)
00:02.0 VGA compatible controller: Intel Corporation Device 3185 (rev 03)
00:0e.0 Audio device: Intel Corporation Device 3198 (rev 03)
00:0f.0 Communication controller: Intel Corporation Device 319a (rev 03)
00:12.0 SATA controller: Intel Corporation Device 31e3 (rev 03)
00:13.0 PCI bridge: Intel Corporation Device 31d8 (rev f3)
00:14.0 PCI bridge: Intel Corporation Device 31d6 (rev f3)
00:14.1 PCI bridge: Intel Corporation Device 31d7 (rev f3)
00:15.0 USB controller: Intel Corporation Device 31a8 (rev 03)
00:17.0 Signal processing controller: Intel Corporation Device 31b4 (rev 03)
00:17.1 Signal processing controller: Intel Corporation Device 31b6 (rev 03)
00:17.2 Signal processing controller: Intel Corporation Device 31b8 (rev 03)
00:17.3 Signal processing controller: Intel Corporation Device 31ba (rev 03)
00:1c.0 SD Host controller: Intel Corporation Device 31cc (rev 03)
00:1f.0 ISA bridge: Intel Corporation Device 31e8 (rev 03)
00:1f.1 SMBus: Intel Corporation Device 31d4 (rev 03)
02:00.0 Ethernet controller: Realtek Semiconductor Co., Ltd. RTL8111/8168/8411 PCI Express Gigabit Ethernet Controller (rev 0c)
03:00.0 Ethernet controller: Realtek Semiconductor Co., Ltd. RTL8111/8168/8411 PCI Express Gigabit Ethernet Controller (rev 0c)
```

Figure 8 – Running benchmarks

This is the “iozone” score of the ODROID-H2 unit #1 in the Ubuntu container.



```
        Iozone: Performance Test of File I/O
                Version $Revision: 3.429 $
                Compiled for 64 bit mode.
                Build: linux-AMD64

        Contributors:William Norcott, Don Capps, Isom Crawford, Kirby Collins
                     Al Slater, Scott Rhine, Mike Wisner, Ken Goss
                     Steve Landherr, Brad Smith, Mark Kelly, Dr. Alain CYR,
                     Randy Dunlap, Mark Montague, Dan Million, Gavin Brebner,
                     Jean-Marc Zucconi, Jeff Blomberg, Benny Halevy, Dave Boone,
                     Erik Habbinga, Kris Strecker, Walter Wong, Joshua Root,
                     Fabrice Bacchella, Zhenghua Xue, Qin Li, Darren Sawyer,
                     Vangel Bojaxhi, Ben England, Vikentsi Lapa.

        Run began: Mon Oct 29 05:57:23 2018

        Include fsync in write timing
        O_DIRECT feature enabled
        Auto Mode
        File size set to 102400 kB
        Record Size 4 kB
        Record Size 16 kB
        Record Size 512 kB
        Record Size 1024 kB
        Record Size 16384 kB
        Command line used: iozone -e -I -a -s 100M -r 4k -r 16k -r 512k -r 1024k -r 16384k -i 0 -i 1 -i 2
        Output is in kBytes/sec
        Time Resolution = 0.000001 seconds.
        Processor cache size set to 1024 kBytes.
        Processor cache line size set to 32 bytes.
        File stride size set to 17 * record size.
                                                              random    random     bkwd    record    stride
              kB  reclen    write  rewrite    read    reread    read     write     read   rewrite      read   fwrite frewrite    fread  freread
          102400       4    71860    90556    93755    94393    44023    88180
          102400      16   178741   231263   219858   245386   110250   232375
          102400     512   453097   440247   481804   494569   459454   491221
          102400    1024   422156   451720   415788   513303   501569   489584
          102400   16384   450103   490871   532380   537491   537374   506306

iozone test complete.




        Iozone: Performance Test of File I/O
                Version $Revision: 3.429 $
                Compiled for 64 bit mode.
                Build: linux-AMD64

        Contributors:William Norcott, Don Capps, Isom Crawford, Kirby Collins
                     Al Slater, Scott Rhine, Mike Wisner, Ken Goss
                     Steve Landherr, Brad Smith, Mark Kelly, Dr. Alain CYR,
                     Randy Dunlap, Mark Montague, Dan Million, Gavin Brebner,
                     Jean-Marc Zucconi, Jeff Blomberg, Benny Halevy, Dave Boone,
                     Erik Habbinga, Kris Strecker, Walter Wong, Joshua Root,
                     Fabrice Bacchella, Zhenghua Xue, Qin Li, Darren Sawyer,
                     Vangel Bojaxhi, Ben England, Vikentsi Lapa.

        Run began: Mon Oct 29 06:02:39 2018

        Include fsync in write timing
        O_DIRECT feature enabled
        Auto Mode
        File size set to 2048000 kB
        Record Size 16384 kB
        Command line used: iozone -e -I -a -s 2000M -r 16384k -i 0 -i 1 -i 2
        Output is in kBytes/sec
        Time Resolution = 0.000001 seconds.
        Processor cache size set to 1024 kBytes.
        Processor cache line size set to 32 bytes.
        File stride size set to 17 * record size.
                                                              random    random     bkwd    record    stride
              kB  reclen    write  rewrite    read    reread    read     write     read   rewrite      read   fwrite frewrite    fread  freread
         2048000   16384   513115   513180   538662   541780   537234   515589

iozone test complete.
```

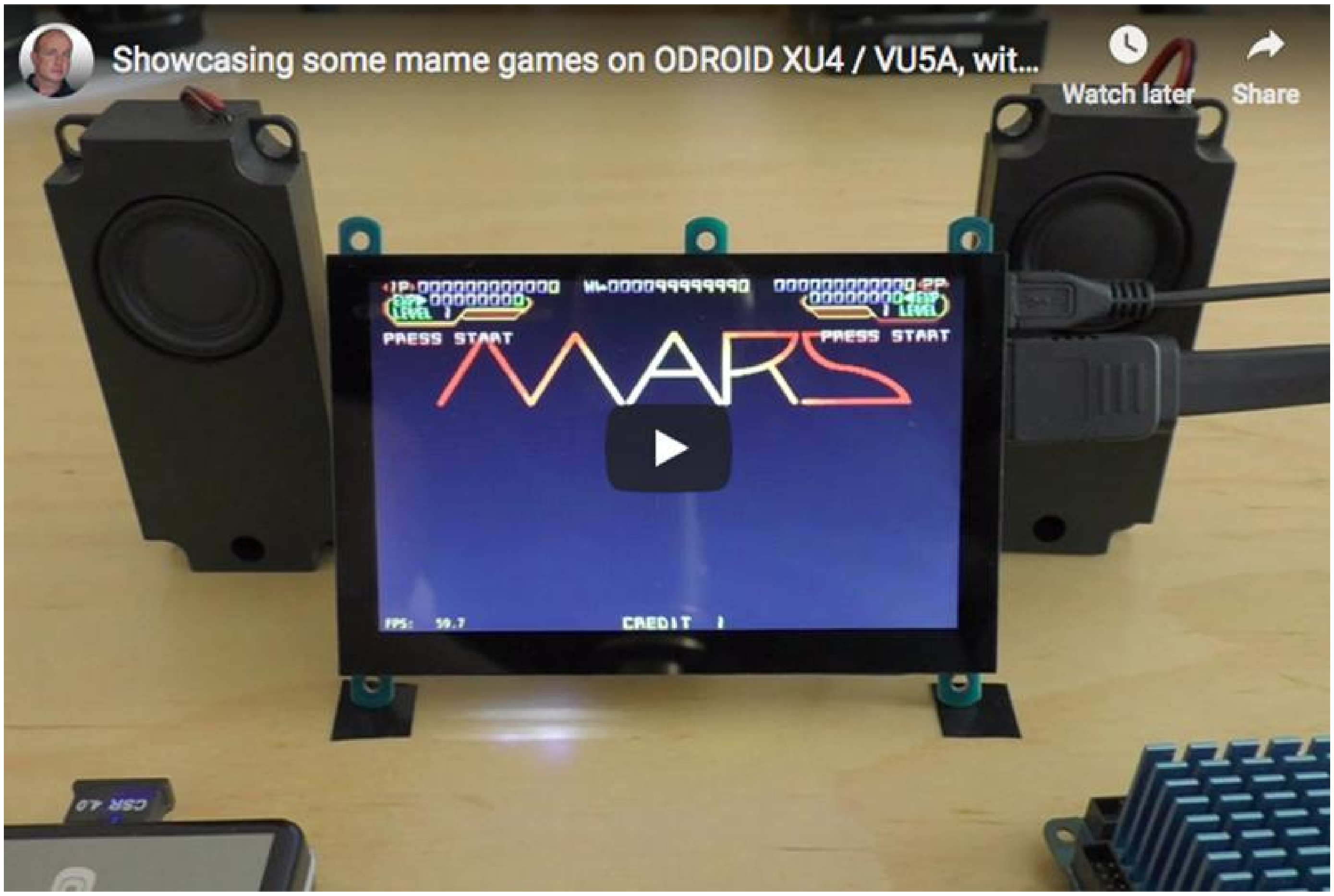
Figure 9 – ODROID-H2 Unit #1 in the Ubuntu container

If you have any issue accessing the machines on ODROID-BENCH or other requests, please refer to the thread at viewtopic.php?f=29&t=32257.

For comments, questions, and suggestions, please visit the original post at https://forum.odroid.com/viewtopic.php?f=29&t=32536.

Building RetroArch

December 1, 2018 By AreaScout Gaming, ODROID-XU4



If you are looking for a frontend to game emulators, you can try RetroArch. It has been ported to the ODROID-XU4 family of Single Board Computers (SBC's). You can follow the steps below to install and use it on your system.

Building and configure RetroArch

We need to obtain the source code, apply a needed patch and build. The small patch basically prevents the display of the menu with a black background.

```
$ git clone
https://github.com/libretro/RetroArch.git
$ cd RetroArch
$ wget -O retro.patch
https://pastebin.com/raw/1SCeb8EG
$ patch -p1 < retro.patch
$ ./configure --enable-opengles3 --enable-
opengles
--enable-neon --enable-floathard --enable-
freetype
```

```
$ make -j7
$ sudo make install
```

Now start it with the command:

```
$ retroarch
```

Apply some useful settings

While you do not have to use the settings listed below, I have included them so you can use them if you wish, as a starting point and tweak them to your preference later.

Update the Assets (Icons, background pictures and stuff), you can find it here:

```
MainMenu -> Online Updater -> Update Assets
```

I recommend you update these packages: **Core Info Files, Joypad Profiles, Database, GLSL Shaders.** You can use the **Core Updater** to get some emulators.

Enable Advanced Settings:

```
Settings -> User Interface -> Show Advanced
Settings -> ON
```

Enable Threaded Video – It will enhance the emulation quite a lot:

```
Settings -> Video -> Threaded Video -> ON
```

Enable FPS counter too. It is helpful to see how fast the emulation runs, especially when you setup things:

```
Settings -> Onscreen Display -> Onscreen
Notifications -> Display Framerate -> ON

Settings -> Onscreen Display -> Onscreen
Notifications -> Show frame count on FPS
Display -> OFF

Settings -> Driver -> Audio Driver ->
alsathread
```

and if you are on VU5A:

```
Settings -> Onscreen Display -> Onscreen
Notifications -> Notification size -> 18

Settings -> Onscreen Display -> Onscreen
Notifications -> Notification X position ->
0.010

Settings -> Onscreen Display -> Onscreen
Notifications -> Notification Y position ->
0.010
```

If you already have games saved in a folder on your ODROID-XU4, you can scan for them:

```
Import Content -> Scan Directory
```

When prompted, you can select the root game folder to let RetroArch scan for your games. They will appear on the right side of the menu after some time.

References

https://www.retroarch.com/
https://forum.odroid.com/viewtopic.php?f=98&t=32173#p233821
https://youtu.be/6Ewgov7_TXM

Meet An ODROIDian: Kamots Tech

December 1, 2018 By Rob Roy Meet an ODROIDian

Please tell us a little about yourself. I live in Florida (aka the Sunshine State), where I was born and raised. I have always lived in Florida because it is warm, there's so much to do, and the IT industry has been growing steadily with a lot of promise on the horizon. I went to college for Computer Networking and, since graduating, I've worked in Information Technologies for over 15 years. I am married and my wife works in Marketing. We have a dog. He is a 9 year old Weimaraner and he has earned many nicknames, including Sir Barks A Lot.

Figure 1 – A Florida sunset

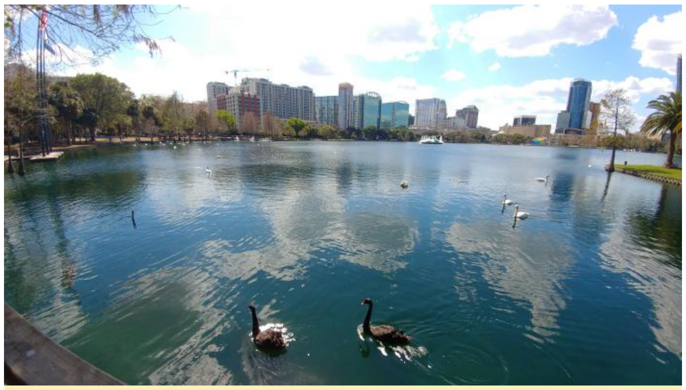


Figure 2 – The Orlando, Florida skyline

Figure 3 – Visiting the wolf preserve in northern Florida

How did you get started with computers? I originally got started in computers based on my interest in electronics when I was young. My father, who is an engineer, was issued a computer by the company he worked for and I was fascinated by it but wasn't allowed to use it since I was well known for taking things apart and only sometimes putting them back together. I saved up my lawn mowing money and eventually bought a used 8088 computer from a friend. It was manufactured by a company called Leading Edge and ran DOS 5.5 from a 40MB hard drive but I thought it was awesome. I learned BASIC then Turbo C on that computer and I still enjoy programming in C variants today. The Internet existed but I mainly connected to BBSes to get files and play games until later on, when I had a better computer and got interested in Linux.

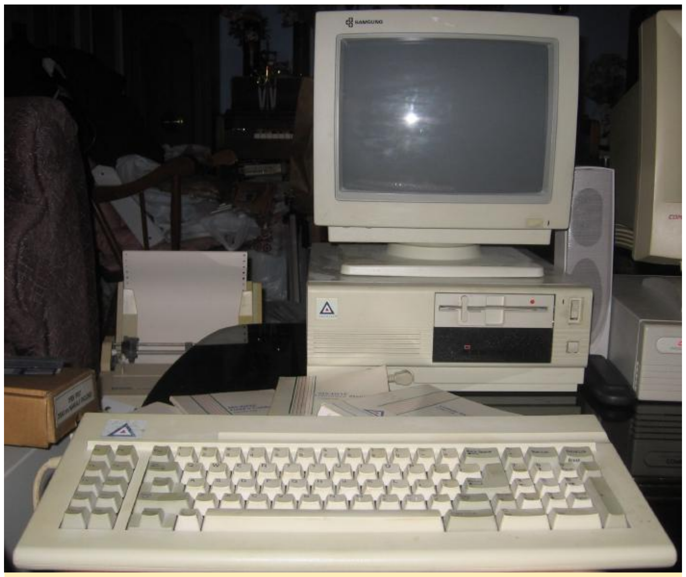


Figure 4 – Kamots got started by using a Leading Edge computer

What attracted you to the ODROID platform? I first heard of the ODROID platform and HardKernel when I read about the ODROID-GO, though I can't recall where I read about it originally. I thought it was cool since it had a lot of capabilities for a good price, and I ordered one right away. I then got involved with the ODROID community and started making YouTube videos about the GO. I've enjoyed helping others discover how to use the emulators and exploring new projects, like attaching a wireless charging system and testing new emulators.

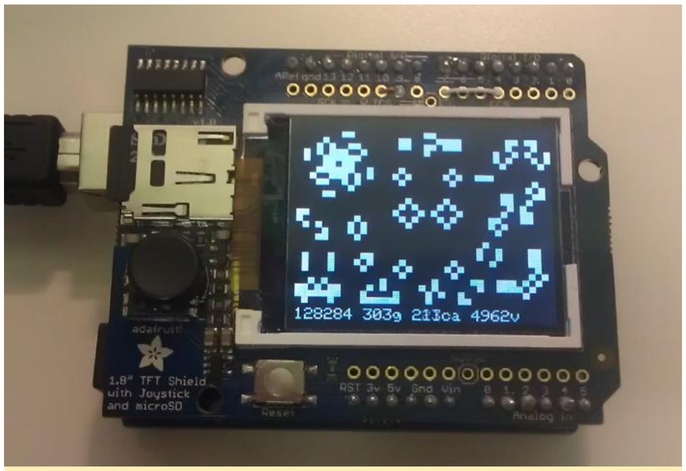


Figure 5 – Conway's Game Of Life on Arduino

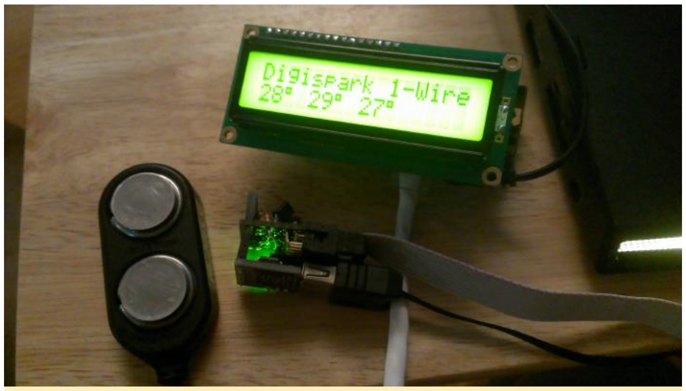


Figure 6 – An early 1-wire temperature sensor project

How do you use your ODROIDs? I run several different emulators on my ODROID-GO and I help out with development as time allows. Lately, I have been enjoying learning the Commodore 64 platform using the new emulator on my GO. I am looking forward to the future additions to this platform.

Which ODROID is your favorite and why? The ODROID-GO. It is just portable fun. It has a touch of nostalgia, which I think is why most everyone gravitates towards it initially, and I'm enjoying the older games all over again.

What innovations would you like to see in future Hardkernel products? I think more products like the ODROID-GO that make electronics fun for everyone would be good. An affordable single-board computer that supports M.2 SSDs would be next on my wish list. I realize that these may be available elsewhere, but I would love to see HardKernel develop a small version around the size of the ODROID-C1+.

What hobbies and interests do you have apart from computers? My wife and I are both SCUBA divers. For me, it is the closest I'll likely get to being in outer space. I follow space science such as the Mars rover missions, the Voyagers, New Horizons, and the International Space Station. I try to watch every launch or major event if my work schedule allows. When I was young, I used to listen to the NASA audio feed during Space Shuttle missions. I have been an Amateur Radio operator for over 20 years and still find it fun to communicate around the world using just an antenna in the backyard. I mainly use digital modes such as JT65 and PSK31 on HF (shortwave) bands. I also enjoy going to the range and putting holes in far away paper with a bow or firearm, but I do not hunt. I like to go geocaching, and travel both with my wife and to visit friends.

Figure 7 – A geocaching trackable item

Figure 8 – The San Francisco Golden Gate Bridge

Figure 9 – A Jamaican beach

Figure 10 – The Space Shuttle Atlantis in the Kennedy Space Center

What advice do you have for someone wanting to learn more about programming? Begin with a device you can program that interacts with the world. When starting out, it can be boring to write a program where the result is only on a screen. However, when someone writes a simple program that achieves something externally, like changing the TV volume or monitoring the weather outside using a remote sensor, I think it makes things more tangible and the imagination begins to see other opportunities in the real world. This can make it more fun for someone just learning and encourage future projects. Everyone learns differently so try different things to see how you learn best and stay inspired to do more. I do recommend eventually learning a C-based language since a lot of programming languages are based on C. It will help you understand many different languages once you understand the basics of C.