

Compilation of the firmware

There are several ways to compile the firmware. As the traditional way, check the original [Building from Source](#) document, however, Docker is recommended because it provides a very clean way to go from source to a .bin file.

[Using Docker and Kitematic](#)

[Docker command-line reference](#)

[Using Buddyworks and other CI platforms](#)

[Notes for Buddy.Works \(and other CI platforms\)](#)

[Using ARM on Debian host](#)

[All in one script for ARM on Debian host](#)

Using Docker Hub and Kitematic

Step 1: Prepare environment

1. Install [Docker](#)
2. Get [Kitematic](#)
3. Install [GitHub Desktop](#)

Step 2: Clone the repository

If you are using Windows, line endings may produce some errors. For example: 'python/r' not found messages are product of a problem with the line endings. This must be done prior to cloning the repository for compilation to succeed. To prevent this, configure git to not manipulate these line endings, open a terminal and execute: `y git config --global core.autocrlf false`

You can also check the current configuration by omitting the `false` at the end of the command.

```
Command Prompt

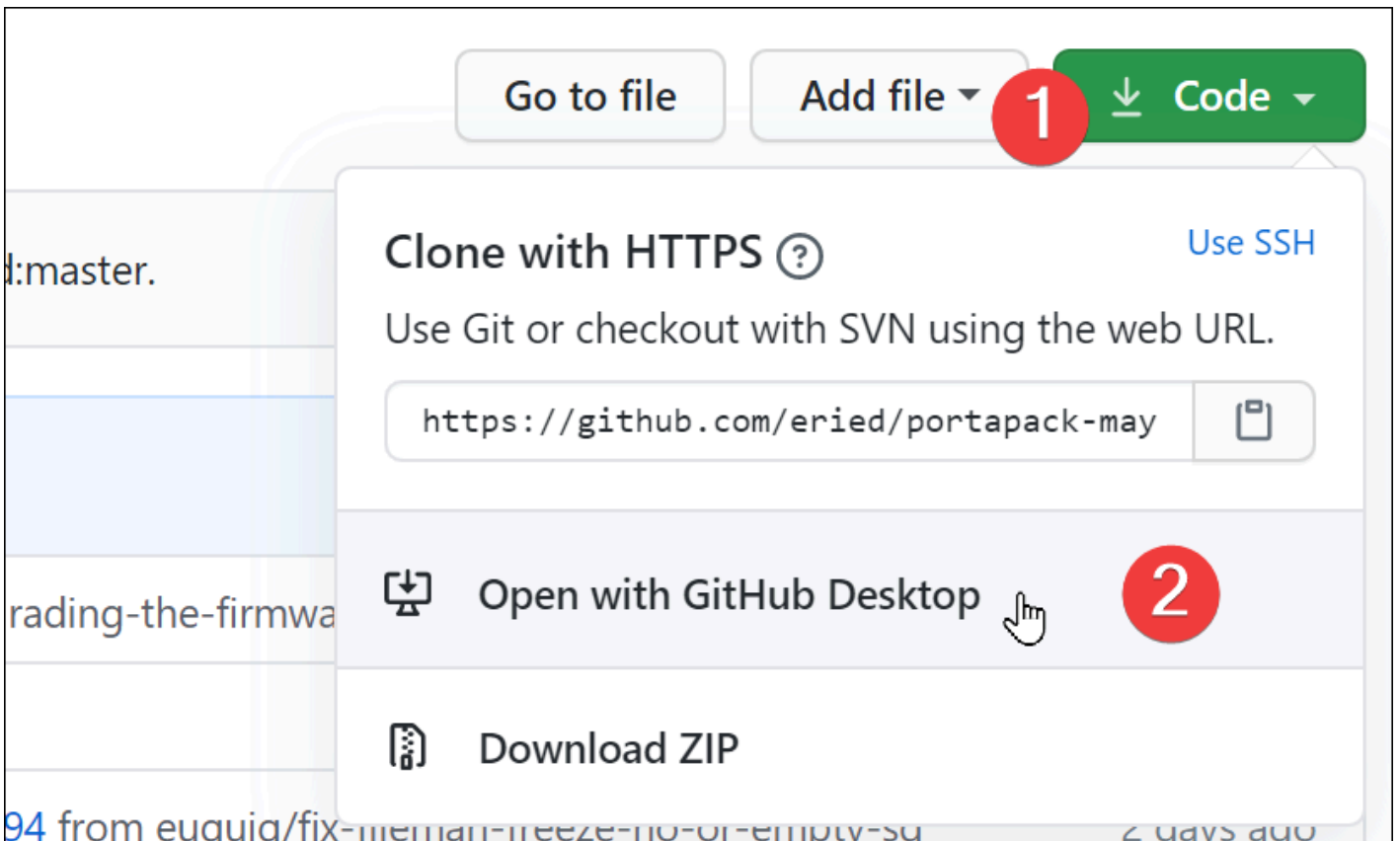
C:\Users\erwin\Documents\GitHub\portapack-mayhem>git config --global core.autocrlf false

C:\Users\erwin\Documents\GitHub\portapack-mayhem>git config --global core.autocrlf
false

C:\Users\erwin\Documents\GitHub\portapack-mayhem>
```

Important: If you want to collaborate to the project, [Fork the repository](#) to your own account and continue this instructions from your own fork.

Open Github Desktop, and click "Open with Github Desktop" from the main page of the repository (or your fork), under the button "Code".



Finally, create a **build** folder inside of the repository. From Github Desktop, just click "Repository / Show in Explorer" and create an empty folder named **build**. This folder will be used for the compilation output.

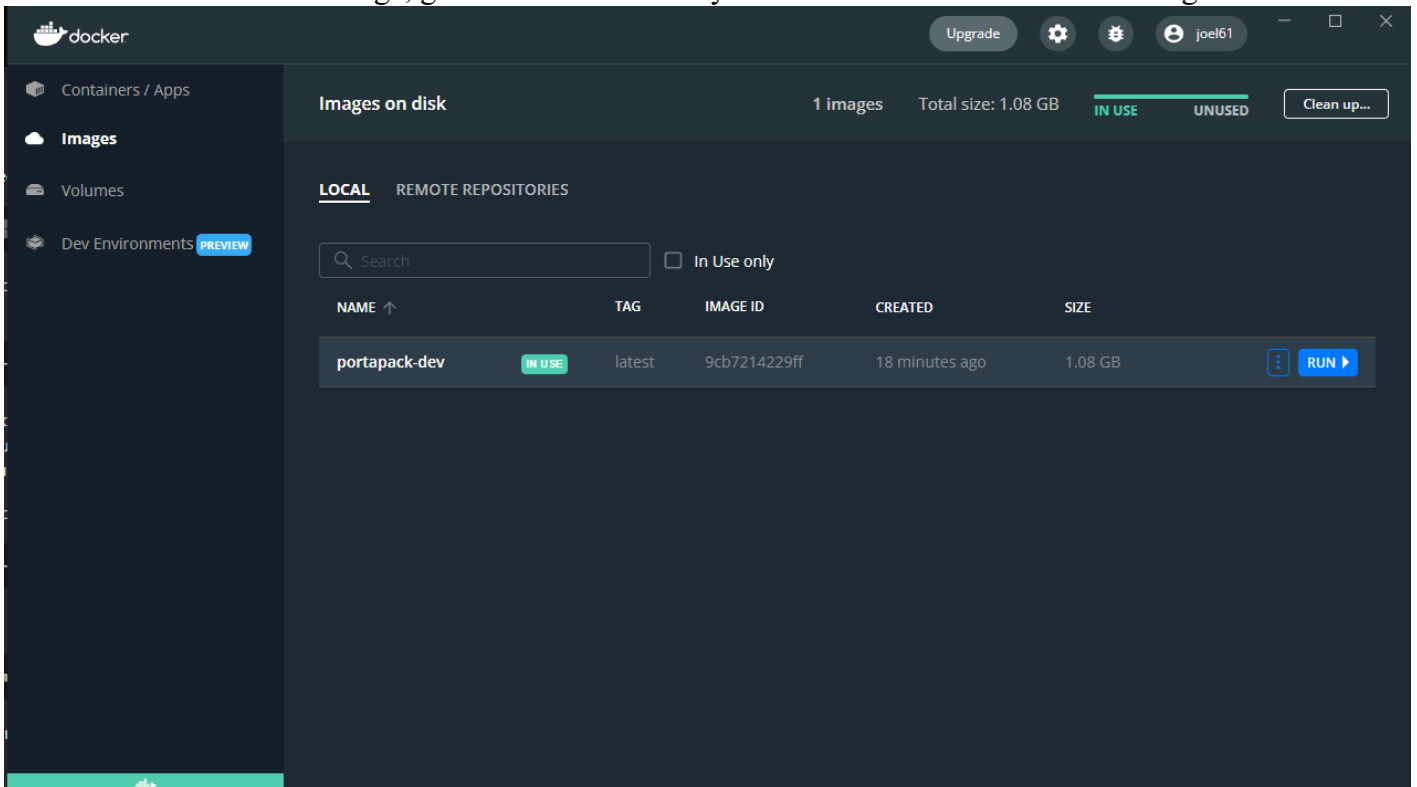
Step 3: Prepare the Docker container

Note: You need to make sure you have also cloned the hackrf folder. to do that run: `git submodule update --init --recursive`

Open up a terminal in the root of the cloned git repo and run: `docker build -t portapack-dev -f dockerfile-nogit .`

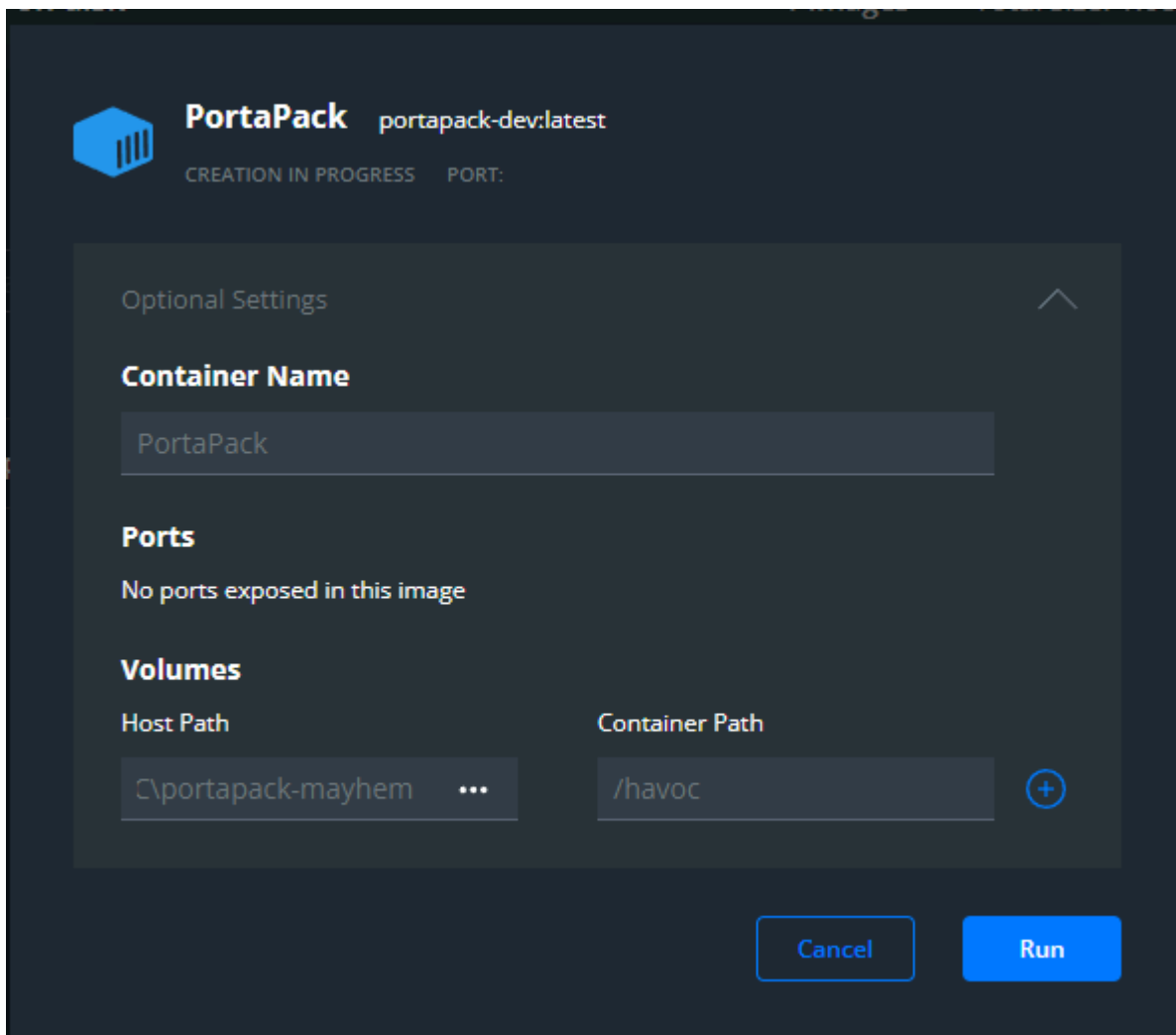
(the image only works on x86 systems, if you are running docker on an arm system, as a workaround, you can get it build and run an amd64 image but it will run in x86 emulation and will be slow. To make the amd64 image use: `docker build --platform linux/amd64 -t portapack-dev -f dockerfile-nogit .`)

After its built the docker image, go back to docker and you should see this screen under images



Click on the blue run button and then click the dropdown to expand Optional Settings.

Make sure they look like this:



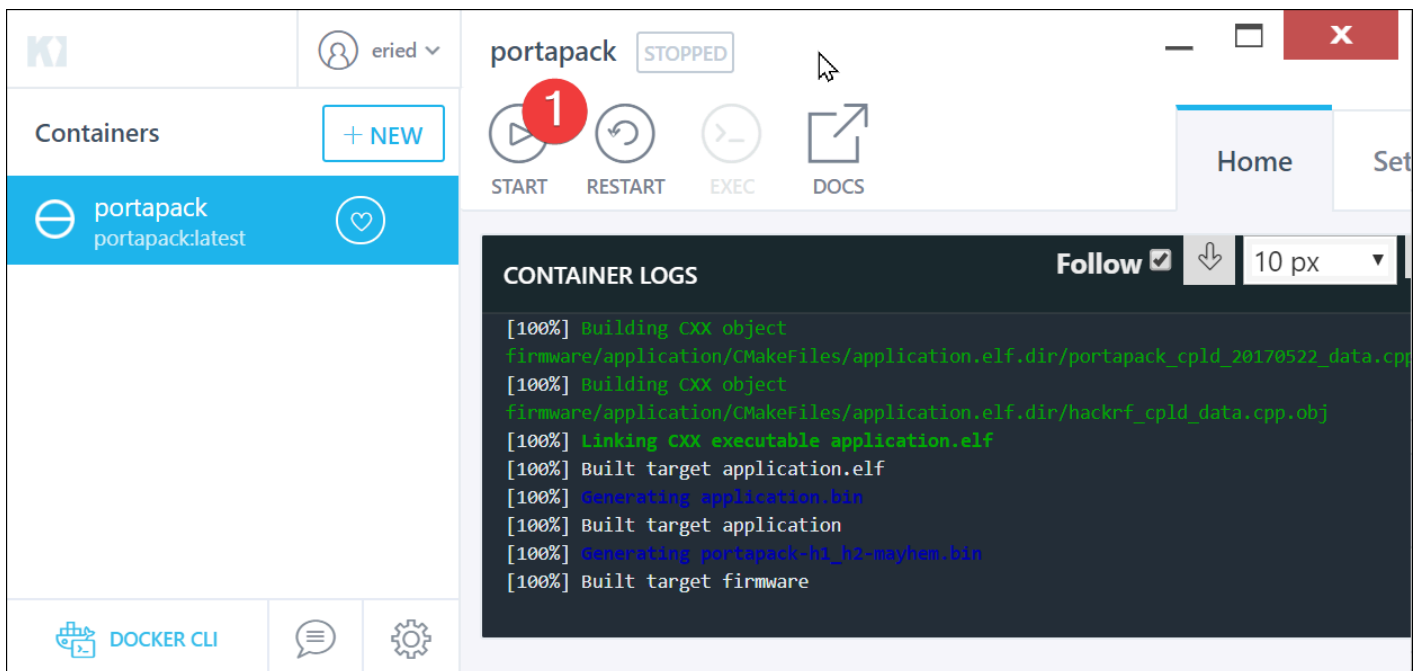
Host path is the root of your repo.

After that click run!

Note: Come across a /usr/bin/env: 'python\r': No such file or directory error? RTFM and go back to step 2 <https://github.com/portapack-mayhem/mayhem-firmware/wiki/Compile-firmware#step-2-clone-the-repository>

Step 4: Compile!

Everytime you run the container you prepared in the previous step, it will compile the source and (if successful) leave the results in build/firmware/



If you have additional questions, please check [this guide](#).

Docker - Command line reference

If you are inclined for using the command line, you can try the following:

- Clone the repository and submodules:

```
git clone https://github.com/portapack-mayhem/mayhem-firmware.git
cd portapack-mayhem
git submodule update --init --recursive
```

- For building the docker image: `docker build -t portapackccache -f dockerfile-nogit .`
- For running the image to build firmware (in the root of the repo): `docker run -it --rm -v ${PWD}:/havoc portapackccache`
This runs and then immediately deletes the container, so that they aren't pile up in your docker instance.
 - You can specify the number of jobs to run in parallel during compilation. To speed up the build specify the number of cores available: `docker run -it --rm -v ${PWD}:/havoc portapackccache -j4`
 - Alternatively if you want to have a single persistent container, and just execute it when necessary
 - Create the persistent container (it will also build the firmware once): `docker run --name portapackbuild -it -v ${PWD}:/havoc portapackccache -j4`
 - Run the existing container: `docker start portapackbuild`

You no longer have to create a `build` folder before running the image.

Using Buddy.Works (and other CI platforms)

You can use the following `_yaml` as your template for your CI platform (pipeline export from [buddy.works](#)):

```
- pipeline: "Build firmware"
  trigger_mode: "ON_EVERY_PUSH"
  ref_name: "master"
  ref_type: "BRANCH"
  auto_clear_cache: true
  trigger_condition: "ALWAYS"
```

```

`actions:`
`- action: "Build Docker image"
  `type: "DOCKERFILE"
  `dockerfile_path: "dockerfile-nogit"
  `do_not_prune_images: true
  `trigger_condition: "ALWAYS"
`- action: "Execute: mkdir build"
  `type: "BUILD"
  `working_directory: "/buddy/portapack-havoc"
  `docker_image_name: "library/ubuntu"
  `docker_image_tag: "18.04"
  `execute_commands:
  `- "mkdir -p build"
  `volume_mappings:
  `- "[:/buddy/portapack-havoc"
  `trigger_condition: "ALWAYS"
  `shell: "BASH"
`- action: "Run Docker Image"
  `type: "RUN_DOCKER_CONTAINER"
  `use_image_from_action: true
  `volume_mappings:
  `- "[:/havoc"
  `trigger_condition: "ALWAYS"
  `shell: "SH"

```

Notes for Buddy.Works (and other CI platforms)

If you decide to [ignore this guide](#) and use the command line instead, you will need to include submodules

```
git clone --recurse-submodules --remote-submodules <url>
```

Using ARM on Debian

- Untested on other linux flavors
- Thanks to @aj#3566 from discord for it
- For convenience the compiler will be installed to /opt/build
- Needed steps
 1. Update a Debian based OS
 2. Install the necessary ARM compiler to /opt/armbin (if not done before)
 3. Link ARM compiler to your bash environment
 4. Clone Mayhem repository from GitHub (if not done before)
 5. Give user permission to the Mayhem repository
 6. Create makefile through cmake and compile
 7. Flash the firmware
- Once done you only need to call 'make' in the /opt/portapack-mayhem/firmware/build directory
- You can speed up the building process by calling 'make -j 8' instead, where 8 is the number of physical CPU cores (if you have some compiling errors to check it's better to call it without '-j 8')
- **Use the following commands while logged into your every day user profile. :)**

1. Update a Debian based OS, install cmake, python, pyyaml

```

sudo apt-get update
sudo apt-get install -y git tar wget dfu-util cmake python3 bzip2 lz4 curl hackrf python3-distutils python3-curl
curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py; python3 get-pip.py
pip install pyyaml

```

2. Install the necessary ARM compiler to /opt/armbin (if not done before)

```
sudo mkdir /opt/build
cd /opt/build
sudo wget -O gcc-arm-none-eabi.tar.bz2 'https://developer.arm.com/-/media/Files/downloads/gnu-rm/9-2020q2/'
sudo mkdir armbin
sudo tar --strip=1 -xjvf gcc-arm-none-eabi.tar.bz2 -C armbin
```

Download an up to date version here if you want to try: <https://developer.arm.com/downloads/-/arm-gnu-toolchains>
The nightly build/release system is using version 9.2.1, known as 9-2019-q4-major, found here: <https://developer.arm.com/nightly-builds>
Don't forget to change the paths or filenames accordingly

3. Link ARM compiler to your bash environment

```
echo 'PATH=/opt/build/armbin/bin:/opt/build/armbin/lib:$PATH' >> ~/.bashrc
source ~/.bashrc
```

4. Clone the eried's mayhem repository from GitHub (if not done before) into /opt

```
cd /opt
sudo git clone --recurse-submodules https://github.com/portapack-mayhem/mayhem-firmware.git
```

5. Give permission for the portapack-mayhem directory to your user

```
sudo chown -R my_user:my_usergroup /opt/portapack-mayhem
```

6. Create makefile through cmake and compile (it's important to call the PATH cmd in step 3 just before making the cmake)

```
cd /opt/portapack-mayhem
mkdir build
cd build
cmake ..
make
```

If you want, use -j argument to increase the compile speed, for example `make -j` to auto decide the numbers of threads to compile, or manually set the thread numbers, for example `make -j4`

Developers wishing to test selected functions in the firmware code by running them on their linux PC can also use the command `make build_tests` and then `ctest --output-on-failure` to run the tests.

7. Flash the firmware to HackRF

```
hackrf_spiflash -w /opt/portapack-mayhem/build/firmware/portapack-h1_h2-mayhem.bin
```

All in one script for ARM on Debian host

If you want to have all these commands in one go, go to <https://github.com/GullCode/compile-flash-mayhem> and download `compile-flash-mayhem.sh` and adjust it to fit your needs

Other Toolsets

- [Linux Aarch64 gcc-arm-none-eabi-9-2020-q2-update-aarch64-linux.tar.bz2](#)
- [Toolsets archive](#)
- [Compilation error after changing toolsets](#)

Compiling Mayhem Firmware on Kali Linux

Tested at 21. Sept. 2024 on Kali 2024.3 kali-rolling
Should work for any Debian based Distribution, see comments.

Download and install dependencies

```
sudo apt update
sudo apt install git tar bzip2 lz4 wget curl cmake python3 python3-setuptools python3-distutils-extra pyth
```

In many distributions the tool `hackrf` is provided, but outdated because of the release policy. Kali is rolling release, so does not follow this restrictions.

If you have a R9 or newer HackRF and need a newer version, I recommended to download and compile it by yourself. Source: <https://github.com/greatscottgadgets/hackrf>

Download and install the ARM toolchain

If you use another Version as 9.2.1, the mayham cmake warns you:

WARNING: Compiler version mismatch, please use the official compiler version 9.2.1 when sharing builds! Current compiler version: 13.2.1

It might compile. From other projects, as Proxmark3 or Chameleon Ultra, we know the `gcc-arm-none-eabi` in version 13 (provided by various distributions) creates a bigger firmware image, which won't fit into the memory. Even if it compiles without error, the firmware file is useless.

```
sudo mkdir /opt/build
sudo chmod $USER:$(id -gn $USER) /opt/build
cd /opt/build
wget https://developer.arm.com/-/media/Files/downloads/gnu-rm/9-2019q4/gcc-arm-none-eabi-9-2019-q4-major->
mkdir armbin
tar --strip=1 -xjvf gcc-arm-none-eabi-9-2019-q4-major-x86_64-linux.tar.bz2 -C armbin
```

I recommend to adjust the user rights at the first possible moment. Work as root as less as possible. If more than one person works in the system, feel free to set the group to something more generic. For example `src` or `staff`.

Source for the newest toolchain: <https://developer.arm.com/downloads/-/arm-gnu-toolchain-downloads>

Source for deprecated toolchain: <https://developer.arm.com/downloads/-/gnu-rm>

Set toolchain as default

```
echo 'PATH=/opt/build/armbin/bin:/opt/build/armbin/lib:$PATH' >> ~/.zshrc
source ~/.zshrc
```

Kali Linux uses [ZSH](#) as default shell. On [Debian](#) is Bash the default shell, so if you are using Debian or changed the default shell, you need to pipe the echo into `~/.bashrc` (or the related [rc file](#))

Get the Firmware from Github

```
cd ~
mkdir git
cd git
git clone https://github.com/portapack-mayhem/mayhem-firmware/ --recurse-submodules
```

Personal Note:

You could work at any place, for example /opt. The [FHS](#) says /usr/src would be the right place for source codes. But this would be in / and most systems I know got roundabout 20 to 50 GB for / and several 100 GB for /home, so I decide to work with git in /home.

Build the firmware

```
cd ~/git/mayhem-firmware
mkdir build
cd build
cmake ..
make clean && make
```

For the first compile `make clean &&` is not necessary. But if the compilation stops of any reason, you need to use `clean`. If nothing is there, it won't hurt.

Compile on WSL with ninja

① Note

ccache is default disabled because it creates issues in particular situation.

If you are aware its pros and cons and you insist enable it, following this guide:

<https://github.com/portapack-mayhem/mayhem-firmware/wiki/Notes-About-ccache>

If you don't know what's this, just use default.

1. Update your system:

```
sudo apt-get update
sudo apt-get upgrade
```

2. Install dependencies:

```
sudo apt-get install -y git tar wget dfu-util cmake python3 ccache bzip2 lib
```

3. Install pip and pyyaml:

```
curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py --break-system-packa
sudo python3 get-pip.py
pip install pyyaml --break-system-packages
```

4. Set up environment variables:

```
export LANG=C.UTF-8
export LC_ALL=C.UTF-8
```

5. Download and install the ARM toolchain:

```
mkdir -p /opt/build
cd /opt/build
wget -O gcc-arm-none-eabi.tar.bz2 "https://developer.arm.com/-/media/Files/d
sudo mkdir armbin
sudo tar --strip=1 -xjvf gcc-arm-none-eabi.tar.bz2 -C armbin
```

6. Add the ARM toolchain to your PATH:

```
echo 'export PATH=$PATH:/opt/build/armbin/bin' >> ~/.bashrc
source ~/.bashrc
```

7. Clone the PortaPack Mayhem repository (if you haven't already):

```
sudo git clone --recurse-submodules https://github.com/portapack-mayhem/mayh
cd mayhem-firmware
sudo chmod -R 777 ~/mayhem-firmware
```

8. Create a build directory and run CMake:

```
mkdir build
cd build
cmake -G Ninja ..
```

9. Build the firmware:

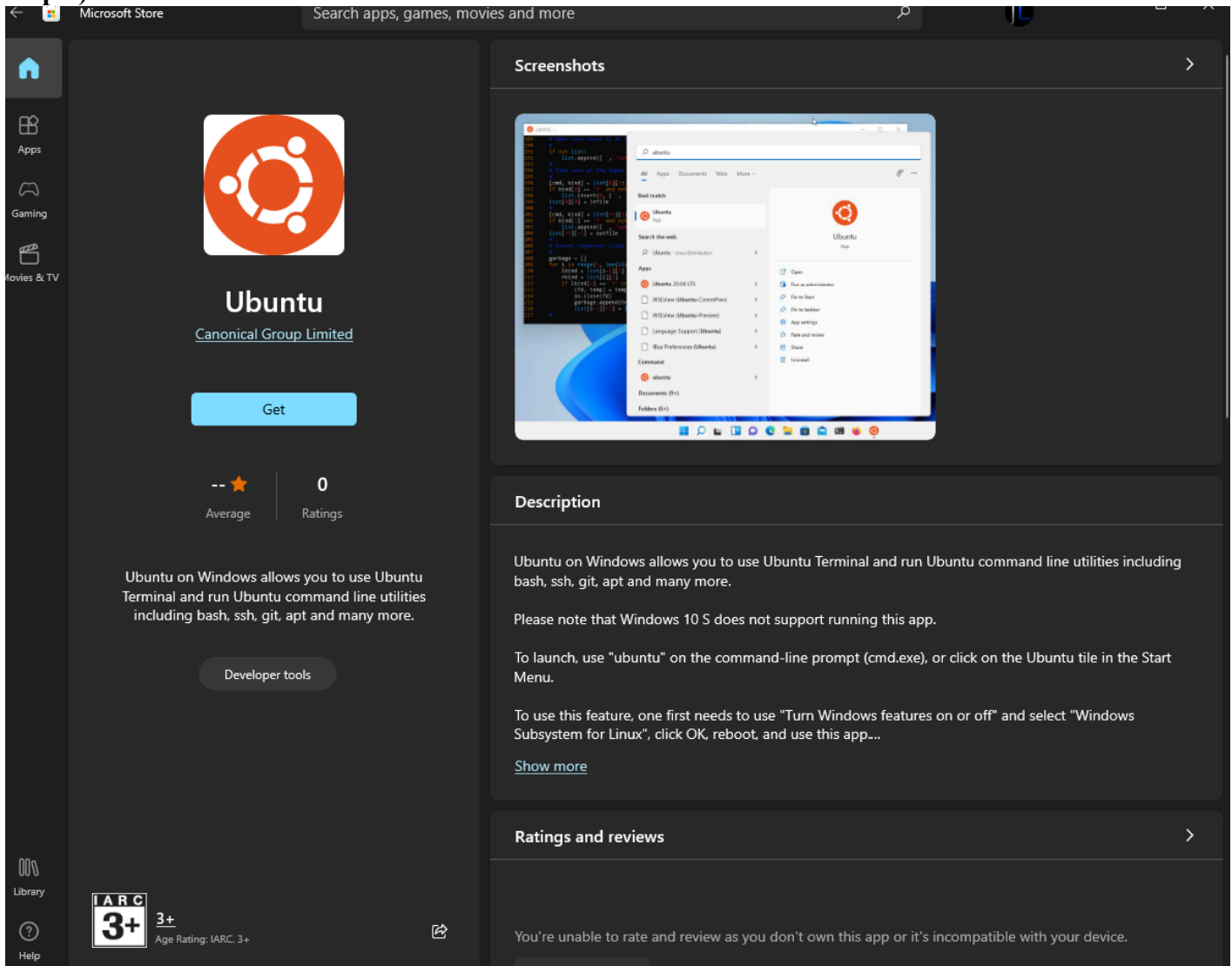
```
ninja
```

How to compile on Windows faster with WSL 2

Note: This is for Windows only

The steps on the [Compile-firmware](#) page work fine for compiling, but you may find it takes around 35 minutes each time. In this guide I will show you how to get that time down to 2 seconds - 1 minute depending on your hardware.

Step 1) Download Ubuntu from the windows store



Step 2) once its opened open up a windows terminal and run `wsl -l -v`

```
A:\Users\jLynx\Documents\Code\C\portapack-mayhem> wsl -l -v
```

NAME	STATE	VERSION
* Ubuntu	Stopped	2

check it says version 2

If it does not, then run: `wsl --set-default-version 2` followed by running `wsl --set-version Ubuntu 2`

Step 3) Once that is done, open the Ubuntu app again (make sure you have set it up so you get to the console in it)

Step 4) Go to docker on windows and go to settings -> resources -> WSL Integration

Step 5) Toggle Ubuntu on (if its not there, click refresh)

Step 6) Open up your windows file explorer and in the address bar go to `\\wsl$\`

Step 7) navigate to `Ubuntu/home/{username}/`

Step 8) copy your portapack-mayhem folder into your dir above

Step 9) Confirm your Ubuntu WSL instance can see it by going `cd ~/portapack-mayhem`

Step 10) If its there then you are good to go. Time to run `docker run -it -v ~/portapack-mayhem:/havoc portapack-dev`

Step 11) Profit! It should be compiling super fast now!

You can then set a network share up to your `\\wsl$\Ubuntu\home\{username}\portapack-mayhem` folder and continue to edit the files from within windows.

Errors While Compiling?

If there were errors in the compiling aspect you can follow these steps.

After Step 5 do this:

Step 5.1) Open Ubuntu

```
spike@TSpiKX-8064: ~  
spike@TSpiKX-8064:~$
```

Step 5.2) Do a `sudo git clone --recurse-submodules` `sudo git clone --recurse-submodules https://github.com/portapack-mayhem/mayhem-firmware.git`

(Note if it still doesn't compile, you might need to run `git submodule update --init --recursive`)

```
spike@TSpiKX-8064: ~  
spike@TSpiKX-8064:~$ sudo git clone --recurse-submodules https://github.com/eried/portapack-mayhem.git  
[sudo] password for spike:  
Cloning into 'portapack-mayhem'...  
remote: Enumerating objects: 23329, done.  
remote: Counting objects: 100% (97/97), done.  
remote: Compressing objects: 100% (46/46), done.  
remote: Total 23329 (delta 52), reused 83 (delta 51), pack-reused 23322  
Receiving objects: 100% (23329/23329), 273.56 MiB | 17.16 MiB/s, done.  
Resolving deltas: 100% (17605/17605), done.  
Updating files: 100% (2984/2984), done.  
Submodule 'hackrf' (https://github.com/mossmann/hackrf.git) registered for path 'hackrf'  
Cloning into '/home/spike/portapack-mayhem/hackrf'...  
remote: Enumerating objects: 17862, done.  
remote: Counting objects: 100% (348/348), done.  
remote: Compressing objects: 100% (153/153), done.  
remote: Total 17862 (delta 206), reused 308 (delta 195), pack-reused 17514  
Receiving objects: 100% (17862/17862), 45.52 MiB | 14.98 MiB/s, done.  
Resolving deltas: 100% (13062/13062), done.  
Submodule path 'hackrf': checked out 'eff4a20022ca5d7f11405c3cdeea6c4195e347d0'  
Submodule 'firmware/libopencm3' (https://github.com/mossmann/libopencm3.git) registered for path 'hackrf/firmware/libopencm3'  
Submodule 'hardware/gsg-kicad-lib' (https://github.com/greatscottgadgets/gsg-kicad-lib.git) registered for path 'hackrf/hardware/gsg-kicad-lib'  
Cloning into '/home/spike/portapack-mayhem/hackrf/firmware/libopencm3'...  
remote: Enumerating objects: 15935, done.  
remote: Counting objects: 100% (4/4), done.  
remote: Compressing objects: 100% (4/4), done.  
remote: Total 15935 (delta 0), reused 4 (delta 0), pack-reused 15931  
Receiving objects: 100% (15935/15935), 3.52 MiB | 6.91 MiB/s, done.  
Resolving deltas: 100% (9594/9594), done.  
Cloning into '/home/spike/portapack-mayhem/hackrf/hardware/gsg-kicad-lib'...  
remote: Enumerating objects: 952, done.  
remote: Counting objects: 100% (208/208), done.  
remote: Compressing objects: 100% (114/114), done.  
remote: Total 952 (delta 140), reused 155 (delta 94), pack-reused 744  
Receiving objects: 100% (952/952), 317.39 KiB | 1.45 MiB/s, done.  
Resolving deltas: 100% (617/617), done.  
Submodule path 'hackrf/firmware/libopencm3': checked out '55021bfff2bc94755059091177f2976e77b1dd7a1'  
Submodule path 'hackrf/hardware/gsg-kicad-lib': checked out '79dbccaef213bb7fb7b0907928e5b359b4618cdb'  
spike@TSpiKX-8064:~$
```

Step 5.3) Give permission for the portapack-mayhem directory to your user

1. Run command `whoami`
2. Run command `sudo chown -R my_user:my_usergroup /home/{username}/portapack-mayhem` (Note: Replace "my_user:my_usergroup" with the return from "whoami")

```
spike@TSpiKX-8064:~$ whoami  
spike  
spike@TSpiKX-8064:~$ sudo chown -R spike:spike /home/spike/portapack-mayhem  
spike@TSpiKX-8064:~$
```

Step 5.4) Create "build" folder

1. Run command `cd portapack-mayhem`
2. Run command `ls` (Note how there's no build folder)
3. Run command `mkdir build`
4. Run command `ls` (Note how a build folder is present)

```

spike@TSpiKX-8064:~$ cd portapack-mayhem
spike@TSpiKX-8064:~/portapack-mayhem$ ls
CMakeLists.txt LICENSE LICENSE.GPL-2.0-or-later LICENSE.md README.md build dockerfile dockerfile-nogit docs firmware flashing hackrf hardware sdcard
spike@TSpiKX-8064:~/portapack-mayhem$ mkdir build
spike@TSpiKX-8064:~/portapack-mayhem$ ls
CMakeLists.txt LICENSE LICENSE.GPL-2.0-or-later LICENSE.md README.md build dockerfile dockerfile-nogit docs firmware flashing hackrf hardware sdcard
spike@TSpiKX-8064:~/portapack-mayhem$

```

Prerequisites for Compiling

Step 5.5) Prevent 'python/r' not found error

1. Run command `git config --global core.autocrlf false`.

Or omit the false to see it's status.

```

spike@TSpiKX-8064:~/portapack-mayhem$ git config --global core.autocrlf false
spike@TSpiKX-8064:~/portapack-mayhem$ git config --global core.autocrlf
false
spike@TSpiKX-8064:~/portapack-mayhem$

```

Compile

Step 5.6)

1. Run command `docker build -t portapack-dev -f dockerfile-nogit .`
2. Run command `docker run -it -v ~/portapack-mayhem:/havoc portapack-dev`

```

spike@TSpiKX-8064:~/portapack-mayhem$ docker build -t portapack-dev1 -f dockerfile-nogit .
[+] Building 3.0s (12/12) FINISHED
=> [internal] load build definition from dockerfile-nogit                                0.3s
=> => transferring dockerfile: 44B                                                    0.0s
=> [internal] load .dockerignore                                                       0.3s
=> => transferring context: 2B                                                         0.0s
=> [internal] load metadata for docker.io/library/ubuntu:xenial                      2.2s
=> [1/8] FROM docker.io/library/ubuntu:xenial@sha256:1f1a2d56de1d604801a9671f301190704c25d604a416f59e03c04f5c6ffee0d6 0.0s
=> CACHED [2/8] WORKDIR /havoc/firmware                                              0.0s
=> CACHED [3/8] RUN apt-get update && apt-get install -y git tar wget dfu-util cmake python3 ccache bzip2 curl && apt-get -qy autoremove 0.0s
=> CACHED [4/8] RUN curl https://bootstrap.pypa.io/pip/3.4/get-pip.py -o get-pip.py && python3 get-pip.py 0.0s
=> CACHED [5/8] RUN pip install pyyaml                                              0.0s
=> CACHED [6/8] RUN ln -s /usr/bin/python3 /usr/bin/python && ln -s /usr/bin/pip3 /usr/bin/pip 0.0s
=> CACHED [7/8] RUN mkdir /opt/build && cd /opt/build && wget -O gcc-arm-none-eabi https://developer.arm.com/-/media/Files/downloads/gnu-rm/9-2019q4/gcc-arm-none-0.0s
=> CACHED [8/8] RUN mkdir ~/bin && cd ~/bin && for tool in gcc g++ cpp c++;do ln -s $(which ccache) arm-none-eabi-$tool;done 0.0s
=> exporting to image                                                                0.3s
=> => exporting layers                                                                0.0s
=> => writing image sha256:7a88ed19019bcae93a932f8da18cc1c51852aae4d4047d059e129dc1e032acd3 0.0s
=> => naming to docker.io/library/portapack-dev1                                     0.0s

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them

```

```

spike@TSpiKX-8064:~/portapack-mayhem$ docker run -it -v ~/portapack-mayhem:/havoc portapack-dev1
-- Cross-compiling with the gcc-arm-embedded toolchain
-- Toolchain prefix: /opt/build/armbin/arm-none-eabi
-- Cross-compiling with the gcc-arm-embedded toolchain
-- Toolchain prefix: /opt/build/armbin/arm-none-eabi
-- The ASM compiler identification is GNU
-- Found assembler: /opt/build/armbin/bin/arm-none-eabi-gcc
-- Configuring done
-- Generating done
-- Build files have been written to: /havoc/build
Scanning dependencies of target baseband_shared
[ 0%] Building C object firmware/baseband/CMakeFiles/baseband_shared.dir/_/chibios/os/ports/GCC/ARMCMx/crt0.c.obj
[ 0%] Building C object firmware/baseband/CMakeFiles/baseband_shared.dir/_/chibios-portapack/os/ports/GCC/ARMCMx/LPC43xx_M4/vectors.c.obj
[ 0%] Building C object firmware/baseband/CMakeFiles/baseband_shared.dir/_/chibios/os/ports/GCC/ARMCMx/chcore.c.obj
[ 1%] Building C object firmware/baseband/CMakeFiles/baseband_shared.dir/_/chibios/os/ports/GCC/ARMCMx/chcore_v7m.c.obj
[ 1%] Building C object firmware/baseband/CMakeFiles/baseband_shared.dir/_/chibios/os/ports/common/ARMCMx/nvic.c.obj
[ 1%] Building C object firmware/baseband/CMakeFiles/baseband_shared.dir/_/chibios/os/kernel/src/chsys.c.obj
[ 1%] Building C object firmware/baseband/CMakeFiles/baseband_shared.dir/_/chibios/os/kernel/src/chdebug.c.obj
[ 1%] Building C object firmware/baseband/CMakeFiles/baseband_shared.dir/_/chibios/os/kernel/src/chlists.c.obj
[ 1%] Building C object firmware/baseband/CMakeFiles/baseband_shared.dir/_/chibios/os/kernel/src/chvt.c.obj
[ 1%] Building C object firmware/baseband/CMakeFiles/baseband_shared.dir/_/chibios/os/kernel/src/chsched.c.obj
[ 2%] Building C object firmware/baseband/CMakeFiles/baseband_shared.dir/_/chibios/os/kernel/src/chthreads.c.obj
[ 2%] Building C object firmware/baseband/CMakeFiles/baseband_shared.dir/_/chibios/os/kernel/src/chdynamic.c.obj

```

Duration of the first compiling / building phase

On my system it took 8 minutes, and 43 seconds to complete, this would vary based on your system's hardware and configurations

```

2022-12-16 08:30:19 -- Cross-compiling with the gcc-arm-embedded toolchain      2022-12-16 08:38:59 [ 98%] Building CXX object firmware/application/CMakeFiles/application.elf.dir/hackrf_cpuid_data.cpp.obj
2022-12-16 08:30:19 -- Toolchain prefix: /opt/build/armbin/arm-none-eabi      2022-12-16 08:39:00 [ 98%] Linking CXX executable application.elf
2022-12-16 08:30:19 -- Cross-compiling with the gcc-arm-embedded toolchain      2022-12-16 08:39:01 [ 98%] Built target application.elf
2022-12-16 08:30:19 -- Toolchain prefix: /opt/build/armbin/arm-none-eabi      2022-12-16 08:39:01 Scanning dependencies of target application
2022-12-16 08:30:20 -- The ASM compiler identification is GNU      2022-12-16 08:39:01 [ 98%] Generating application.bin
2022-12-16 08:30:20 -- Found assembler: /opt/build/armbin/bin/arm-none-eabi-gcc      2022-12-16 08:39:01 [ 98%] Built target application
2022-12-16 08:30:20 -- Configuring done      2022-12-16 08:39:01 Scanning dependencies of target firmware
2022-12-16 08:30:21 -- Generating done      2022-12-16 08:39:02 [100%] Generating portapack-h1_h2-mayhem.bin
2022-12-16 08:30:21 -- Build files have been written to: /havoc/build      2022-12-16 08:39:02 [100%] Built target firmware
2022-12-16 08:30:22 Scanning dependencies of target baseband_shared

```

Duration of the second compiling / building phase

On the second compile, I made changes on the "ui_geomap.cpp" & "ui_geomap.hpp". These were no more than 25 lines of changes made collectively. And this was compiled in 3 seconds on my system.

```
2022-12-16 08:53:36 -- Cross-compiling with the gcc-arm-embedded toolchain 2022-12-16 08:53:39 [ 56%] Built target hackrf_usb.dfu
2022-12-16 08:53:36 -- Toolchain prefix: /opt/build/armbin/arm-none-eabi 2022-12-16 08:53:39 [ 98%] Built target application.elf
2022-12-16 08:53:36 -- Configuring done 2022-12-16 08:53:39 [ 98%] Built target application
2022-12-16 08:53:37 -- Generating done 2022-12-16 08:53:39 [100%] Built target firmware
2022-12-16 08:53:37 -- Build files have been written to: /havoc/build
2022-12-16 08:53:38 [ 18%] Built target baseband_shared
```

More Info on WSL 2?

Please refer to these official links from Microsoft:

1. <https://learn.microsoft.com/en-us/windows/wsl/install-manual#step-3---enable-virtual-machine-feature>
2. <https://learn.microsoft.com/en-us/windows/wsl/troubleshooting>

Compile on Arch based distro (exclude Asahi)

❗Note

ccache is default disabled because it creates issues in particular situation.

If you are aware its pros and cons and you insist enable it, following this guide:

<https://github.com/portapack-mayhem/mayhem-firmware/wiki/Notes-About-ccache>

If you don't know what's this, just use default.

This guide not works on Asahi, please make sure you are on x86_64 platform

1. Install dependence

```
sudo pacman -S git tar wget dfu-util cmake make python3 bzip2 lz4 curl hackrf
python-distutils-extra python-setuptools python-pip python-yaml
```

Check the output and make sure the packages listed above were installed correctly.

2. Install ARM gcc-arm-none-eabi package from AUR

you can follow the instructions in [Debian based distro page in this wiki](#) as well. This doesn't work before and it turns out that it's nushell caused the bug. If you don't use nushell, you are all good with that set up.

This will automatically add the binaries of arm toolchain to the /usr/bin of your system. Note that you are adding an old toolchain into your system.

1. Go to [the page of gcc-arm-none-eabi package in AUR](#).
2. Click View Changes to check the commit history of this AUR package.
3. Click [version 9-2020-q2](#) to check the specific version of gcc-arm-none-eabi, then click Download to download the package for makepkg.
4. Assuming you download the package to ~.
5. Create a directory to satisfying and checking the package. mkdir AUR
6. mv aur-11b618acbed084c37cdf1568a1bc2b05152af7e1.tar.gz ./AUR
7. cd AUR
8. tar -xvf aur-11b618acbed084c37cdf1568a1bc2b05152af7e1.tar.gz
9. cd aur-11b618acbed084c37cdf1568a1bc2b05152af7e1
10. makepkg

(ARM already fixed this but in case of it happened again we'll leave it here.)

Note that since the SSL certificate of the file this makepkg pointed to already expired, the curl wouldn't download it correctly, thus, you have to add -k argument to your makepkg.conf:

```

sudo vim /etc/makepkg.conf
Edit the line
'https://usr/bin/curl -qgb "" -fLC ---retry 3 ---retry-delay 3 -o %u'
to
'https://usr/bin/curl -qgb "" -fLC ---retry 3 ---retry-delay 3 -k -o
%o %u'
(You may change it back after installing, if you needed)
Then makepkg, waiting it finished.

```

11. Install the package with pacman:

```
sudo pacman -U gcc-arm-none-eabi-bin-9_2020_q2_update-1-x86_64.pkg.tar.zst
```

3. Clone your repo to local and satisfying the sub-module

```

cd ~
git clone https://github.com/portapack-mayhem/mayhem-firmware.git
cd portapack-mayhem
git submodule update --init --recursive

```

4. Give permission to the repo directory in your local and compile it

No need to do this step if all the things you do are on one user.

```

sudo chown -R my_user:my_usergroup ~/mayhem-firmware
cd ~/mayhem-firmware
mkdir build
cd build
cmake ..
make firmware

```

If you want, use `-j` argument to increase the compile speed, for example `make -j firmware` to auto decide the number of threads to compile, or manually set the thread numbers, for example `make -j4 firmware`

Notes

1. You cannot directly install `gcc-arm-none-eabi` from AUR using yay or others tool, otherwise the version would be not match.
2. (ARM already fixed this but in case of it happend again we'll leave it here.)

~~After installing gcc-arm-none-eabi you may change the makepkg.conf back:~~

```

sudo vim /etc/makepkg.conf
Edit the line
'https://usr/bin/curl -qgb "" -fLC ---retry 3 ---retry-delay 3 -k -o
%o %u'
to
'https://usr/bin/curl -qgb "" -fLC ---retry 3 ---retry-delay 3 -o %o
%u'

```

Dev build versions

We are trying to work out what is the most recent version we can update to while making it so all devs can still develop using the version on their OS/distro.

Developer testing with various GCC compiler and hardware versions demonstrates code stability, and helps suss out intermittent issues such as timing issues or uninitialized memory. But, particularly if the nightly

build system uses a different GCC version from most developers, a developer-testing phase may be needed for future firmware release candidates before the worldwide release announcement.

Note that higher GCC versions contain larger libraries that consume more of our limited ROM space.

Dev	GCC Version	Platform	Issues
Stable/nightly	9.2.1	ubuntu:20.04	ADSB fails in v1.8.0 (some claim other versions too TBD)
@jlynx	9.4.0	Ubuntu (WSL)	None (That I am aware of)
@u-foka	13.2.1	M1 Mac	File size too big
@notherngineer	9.3.1	Debian 11	None
@bernd-herzog	9.2.1	ubuntu:22	None
@zxkmm	13.2.0	Arch Linux on X86_64 PC (Intel Core i7-8Gen* ^h)	Image size too big
@zxkmm	13.2.0	Manjaro Linux on X86_64 PC (Intel Core i3-3Gen)	Image size too big
@zxkmm	10.3-2021.10-aarch64	Armbian on Amlogic S905D Cortex-A53 microchip	Image size too big

Notes About ccache

background

ccache is a nice tool, but we certainly realized some issue of its behavior in this project, that could waste your time instead of save your time. So it's **default off** now, unless you manually enable it.

issue that could appear if you enable it

Just keep in mind that these issues could be made:

- When you compiled with ccache enabled, and use git to merge/pull new changes, it can only detect a part of your change, and would compile fail. It cannot be fixed by `make clean`, you have to delete the build dir and start over.
- Sometimes ccache cause linking stage took extremely long.
- Unknown linking error until you `make clean` and start over again.

I want enable it anyway

If you are aware of these issue and you are sure you are able to realized them, when they appear, follow this to enable ccache:

for make:

turn on

```
#(...other steps that in compile guide...)
mkdir build
cd build
cmake -DUSE_CCACHE=ON ..
make -j
#(...other steps that in compile guide...)
```

turn off

```
#(...other steps that in compile guide...)
mkdir build
cd build
cmake -DUSE_CCACHE=OFF ..
```



```
make -j
#(...other steps that in compile guide...)
```

default behavior (off) (if you turned it on, this will become on until you turn it off again)

```
#(...other steps that in compile guide...)
mkdir build
cd build
cmake ..
make -j
#(...other steps that in compile guide...)
```

for ninja:

turn on

```
#(...other steps that in compile guide...)
mkdir build
cd build
cmake -G Ninja -DUSE_CCACHE=ON ..
ninja
#(...other steps that in compile guide...)
```

turn off

```
#(...other steps that in compile guide...)
mkdir build
cd build
cmake -G Ninja -DUSE_CCACHE=OFF ..
ninja
#(...other steps that in compile guide...)
```

default behavior (off) (if you turned it on, this will become on until you turn it off again)

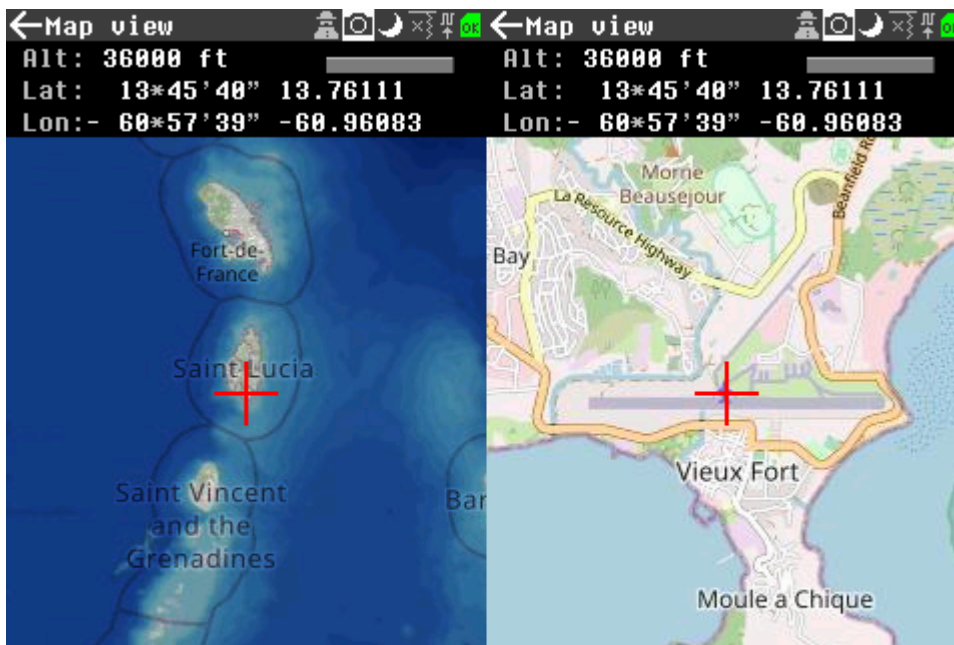
```
#(...other steps that in compile guide...)
mkdir build
cd build
cmake -G Ninja ..
ninja
#(...other steps that in compile guide...)
```

Create a custom map

This is guide to create a custom map for your PortaPack and NOT a custom map with special details.

Who is this for?

This may be useful to you if you find your area/region to be small or unclear.

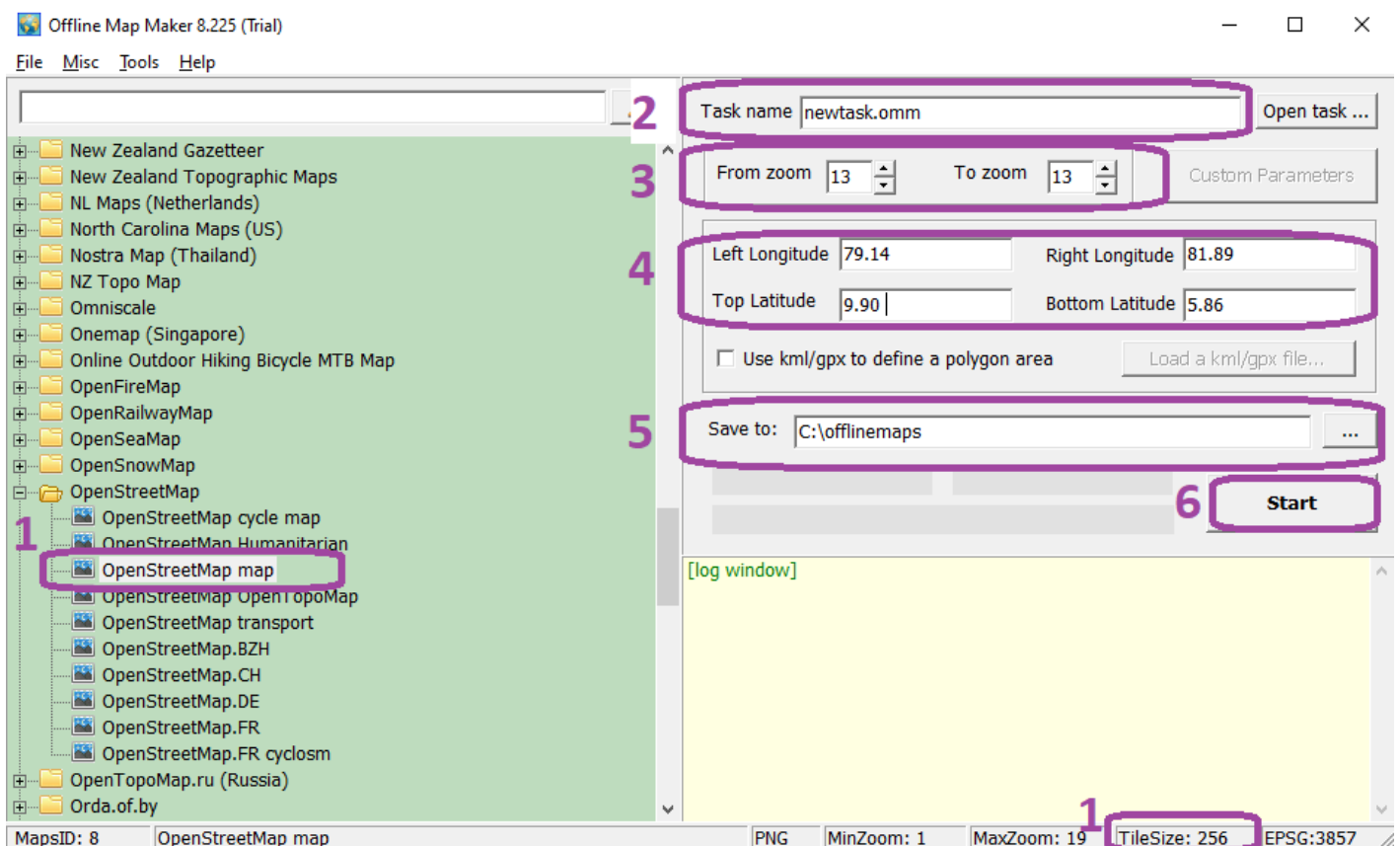


Before

After

Steps to create this custom map using Offline Map Maker :

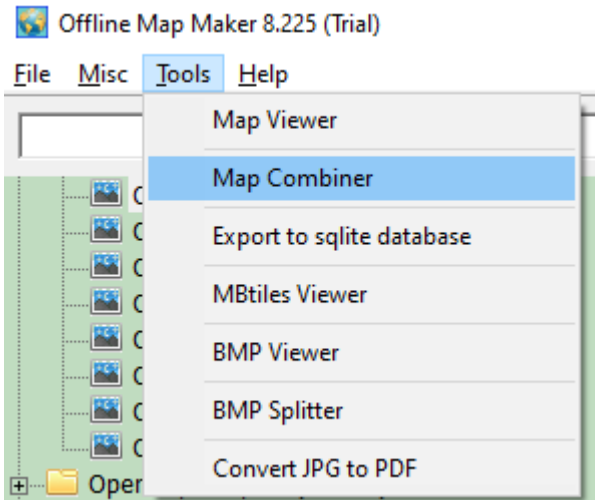
1. Download Offline Map Maker via it's official website: <https://www.allmapsoft.com/omm/index.html> or by clicking this [link](#)
2. After running the install you would do the following:



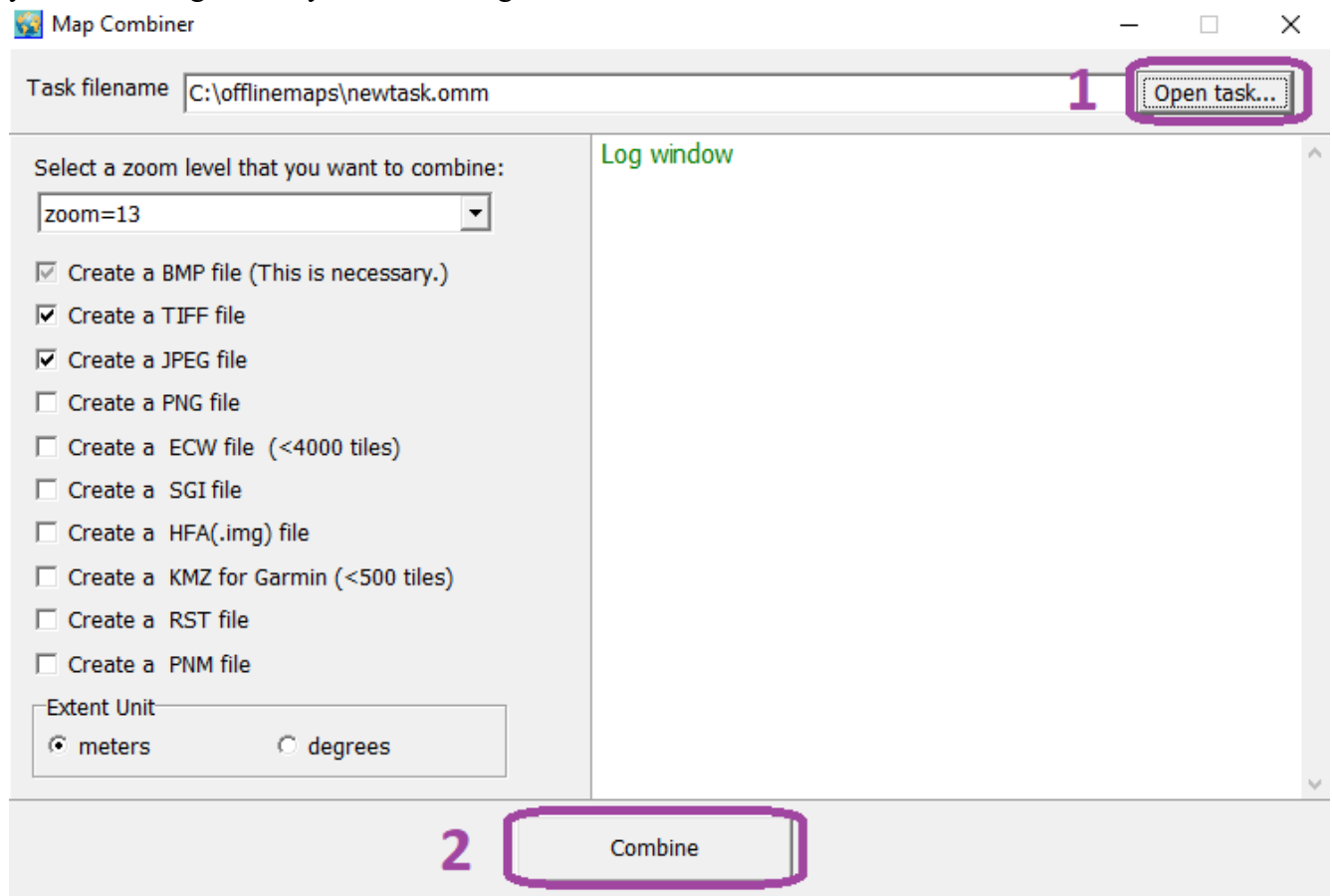
1. Select a Maps type in the left tree (Select maps with only 256 px, 256 px is the tile size. If you not seeing the 256 px, chances are that those without a px size is 256 by default. You can always check with [this site](#) if it's not shown on the GUI)
2. Enter a task name, such like "newtask.omm".

3. Define the zoom level scope. (Since we want the best clarity we would go with the highest zoom which is 13 in the free version, paid is 19. But please note that higher zoom means bigger file size)
4. directly input 4 parameters (Left Longitude, Right Longitude, Top Latitude, Bottom Latitude) to define the area scope of images that you want to download See [here](#) for uploading a KML file type)
5. Select a path to save the project and downloaded images.
6. Then click button "Start".

Combine Downloaded Tiles/Images



7. After having a successful download, you would want to navigate to "Tools"-->"Map Combiner" where you would be greeted by a similar image:



1. Open the location of where your file was downloaded (Ensure "Create a JPEG file" is checked and your selected zoom should appear by default if not, do select).
2. Then click "Combine"

For a more detailed set of instructions visit the official [website](#)

Lastly

Upon the successful downloading and combining of your custom map refer to the following steps:

1. Rename your combined image to world_map.jpg
2. Place the input map in: sdcard/ADSB/world_map.jpg
3. Then run firmware/tools/generate_world_map.bin.py. Where your map will be generated on: sdcard/ADSB/world_map.bin
4. Place the generated .bin file in the respective folder of your SD Card (ADSB/world_map.bin)

Keynotes

- Ensure you have Python installed (I have Python 3.10)
- And the package "Pillow" installed (`pip install Pillow`)

Coding

To see your actual custom map, you need to do a change/edit of code. This change would occur in line 197 of file ui_geomap.cpp which is located at /firmware/application/ui/ or [here](#) for ease of reference. The function that's being changed is `void GeoMap::move(const float lon, const float lat)`.

What I replaced this function with is as follow:

```
` void GeoMap::move(const float lon, const float lat) {
```

```
lon_ = lon;
lat_ = lat;

// Enter Map Coordinates Accordingly
double mapLeftLon = 79.14;
double mapRightLon = 81.89;

double mapLonDelta = mapRightLon - mapLeftLon;

double mapBottomLat = 5.86;

double mapBottomLatDegree = mapBottomLat * pi / 180;

Rect map_rect = screen_rect();

// Using WGS 84/Pseudo-Mercator projection
// x_pos = map_width * (lon_+180)/360 - (map_rect.width() / 2);
x_pos = (lon_ - mapLeftLon) * (map_width / mapLonDelta) - (map_rect.width() / 2);

// Latitude calculation based on https://stackoverflow.com/a/10401734/2278659
// double map_bottom = sin(-85.05 * pi / 180); // Map bitmap only goes from about -85 to 85 lat
// double lat_rad = sin(lat * pi / 180);
double lat_rad = lat_ * pi / 180;
// double map_world_lon = map_width / (2 * pi);
double map_world_lon = ((map_width / mapLonDelta) * 360) / (2 * pi);
// double map_offset = (map_world_lon / 2 * log((1 + map_bottom) / (1 - map_bottom)));
double map_offset = (map_world_lon / 2 * log((1 + sin(mapBottomLatDegree)) / (1 - sin(mapBottomLatDegree))));
// y_pos = map_height - ((map_world_lon / 2 * log((1 + lat_rad) / (1 - lat_rad))) - map_offset) - 128; //
y_pos = map_height - ((map_world_lon / 2 * log((1 + sin(lat_rad)) / (1 - sin(lat_rad)))) - map_offset);

// Cap position
if (x_pos > (map_width - map_rect.width()))
    x_pos = map_width - map_rect.width();
if (y_pos > (map_height + map_rect.height()))
    y_pos = map_height - map_rect.height();
```

```
}
```

What ever code is commented out was the original code. I tried my best to make this plug and go but I'm certain that, that will not be the case for some. In my testing with this code that I've uploaded here, It seems that my projections are off by 2-300 ft. Which is not bad since I couldn't even differentiate if the calculations were even off at the beginning.

Breakdown of Code

I also tried my best to make the code input variables self explanatory. But for those who are new and those who willing to give this code a shot, all that is required of you is to:

1. Enter your mapLeftLon, which is your "Left Longitude"
2. Enter your mapRightLon, which is your "Right Longitude"
3. And enter your mapBottomLat, which is your "Bottom Latitude" These values are the ones you entered / KML uploaded at step 4 at section "Steps to create this custom map using Offline Map Maker". Also Note if your KML file has a lot of decimal points you should enter those exactly since I never tested a round off value but that doesn't mean you should too.

Code formatting

For this we are using clang format version 13. All the config for this is in the .clang-format file. Your IDE should automatically pick up on this when you format your code.

VS Code

Make sure you have C/C++ extension by Microsoft installed and it should automatically pick up your .clang-format file. If not, you can go to the extension settings and specify the location under the Clang_format_style setting

CLI

To format using CLI, make sure you have clang-format-13 installed

```
wget https://apt.llvm.org/llvm.sh
chmod +x llvm.sh
sudo ./llvm.sh 13
sudo apt install clang-format-13
```

Then you can run

```
find firmware/common firmware/baseband firmware/application -iname '*.h' -o -iname '*.hpp' -o -iname '*.cpp' -o -iname '*.c' | xargs clang-format-13
```

or individually:

```
find firmware/common -iname '*.h' -o -iname '*.hpp' -o -iname '*.cpp' -o -iname '*.c' | xargs clang-format-13
find firmware/baseband -iname '*.h' -o -iname '*.hpp' -o -iname '*.cpp' -o -iname '*.c' | xargs clang-format-13
find firmware/application -iname '*.h' -o -iname '*.hpp' -o -iname '*.cpp' -o -iname '*.c' | xargs clang-format-13
```

CLion

CLion has built in clang-format, just press Ctrl + Shift + L (Windows and Linux) to format the code with .clang-format configure file within project directory. If you have clang-tidy or so enabled, you need to

disable them.

Note: KDE default keymap is Alt + Shift + L

PR-process

In order to get your PR merged into the code base, there are a few checks that need to happen first.

- At least one code approver needs to approve your code.
- If any code reviewer left PR comments, then all those PR comments left by reviewers need to be marked as resolved.
- All 3 code format gate checks need to complete successfully. (don't know how to format your code? Check out how to do it here <https://github.com/portapack-mayhem/mayhem-firmware/wiki/Code-formatting>)

Once these steps are all complete, if you are an approved contributor you will notice you have a green merge button available, then means you are good to go and merge your code into the code base. Everyone else will see that tests have passed and will need to ask an approved contributor to merge it in for them (You can as the dev that approved the changes).

If any of these steps are not complete or failed, then you will be blocked from merging your code until they have all been completed.

Username on the nightly change log

A new build of 'next' is released every night. Your contribution will be added to the list of changes that were accepted that day. In order for your user account to display correctly you must either. Enable 'Keep my email addresses private' which will add a unique email address like '417283+YourGitHubUser@noreply...'. Your username will be extracted from the private no-reply email address.

At First Glance

HackRF

The **original HackRF** code can be found at `/hackrf`, it is a subproject inside the source code. It points to <https://github.com/mossmann/hackrf>

Portapack

The rest of the code, including the **portapack** and **havoc/mayhem** extras can be found at `/firmware`

The portapack <--> HackRF interface functions are located in `/firmware/baseband`

There is an underlying OS framework, the **CHIBIOS/RT**, which is located at `/firmware/chibios` for the abstract functionality, and `/firmware/chibios-portapack` for the specific portapack board hardware drivers and configurations.

Application folder

The portapack application software, can be found at `/firmware/application`

User interface: `/firmware/application/ui`

Apps (options in each menu): `/firmware/application/apps`

IC / hardware components interface: `/firmware/application/hw`

Common protocols functionality: `/firmware/application/protocols`

User Interface

Menus and Widgets

MOST menus (excluding OPTIONS and DEBUG, which are actually apps) and **pop-up boxes** (yes/no, yes/cancel, ok) are defined in /firmware/application/ui_navigation.cpp

The **Options menu** is actually an "app" defined at /firmware/application/apps/ui_settings.cpp

The **Debug menu** (also an app) is defined at /firmware/application/apps/ui_debug.cpp

The widgets used in the UI, as **buttons, texts, checkboxes, input fields**, etc. are to be found at /firmware/common/ui_widget.cpp

Boot Process

The boot process is a bit of madness, but justifiable madness.

Overview

The LPC4320 bootloader initializes the Cortex-M4F core to boot from the start of external SPI flash. The M0 core stays in reset. The bootstrap code runs from SPI flash, on the Cortex-M4F. The bootstrap initializes the Cortex-M0 to execute the application code from SPI flash, then sleeps. The application code copies the baseband code into RAM, configures the Cortex-M4F to run from RAM, then resets the Cortex-M4F to begin baseband execution.

(TODO: A diagram would be helpful, showing the M4F and M0 activities vs. time.)

Bootstrap

In the PortaPack image, the Cortex-M4F "bootstrap" image is located at the start of SPI flash. It's executed by the LPC4320 built-in bootloader. The M4 clock is already set to 96MHz. The bootstrap code configures SPIFI to run at (approximately) maximum speed. Then, it initializes the Cortex-M0's memory map to point at the "application" image in SPI flash, and releases the Cortex-M0 from reset. The bootstrap then sleeps the Cortex-M4 until the M0 application needs to run baseband firmware on it.

Application

On the Cortex-M0 core, boot time looks like this:

```
ResetHandler:
  Initialize process stack pointer
  Initialize stack RAM regions (fill with pattern)
  __early_init()
    Enable extra processor exceptions for debugging
  Init data segment (copy SPI flash -> data region in RAM)
  Initialize BSS (fill RAM region with 0)
  __late_init()
    reset()
      Reset most peripherals -- not SCU, SPIFI, or M0APP
    halInit()
      hal_lld_init()
        Init timer 3 as cycle counter (no DWT on M0)
        Init RIT as SysTick (no SysTick on M0)
      palInit()
      gptInit()
      i2cInit()
      sdcInit()
      spiInit()
      rtcInit()
      boardInit()
    chSysInit()
      port_init()
      _scheduler_init()
      _vt_init()
      _core_init()
      _heap_init()
      chSysEnable()
      chThdCreateStatic(_idle_thread_wa, ...)
Constructors
main()
```

```
Destructors
_default_exit()
while(1);
```

Baseband

On the Cortex-M4F core, these are the stages a baseband image moves through:

```
ResetHandler:
    Initialize process stack pointer
    Initialize FPU
    Initialize stack RAM regions (fill with pattern)
    __early_init()
        Enable extra processor exceptions for debugging
    Init data segment (copy SPI flash -> data region in RAM)
    Initialize BSS (fill RAM region with 0)
    __late_init()
        halInit()
            hal_lld_init()
                Init SysTick
                Init DWT as cycle counter
                # Baseband controls no hardware, so no hardware init here.
            boardInit()
        chSysInit()
            port_init()
            _scheduler_init()
            _vt_init()
            _core_init()
            _heap_init()
            chSysEnable()
            chThdCreateStatic(_idle_thread_wa, ...)
Constructors
main()
Destructors
_default_exit()
while(1);
```

Original Wiki by sharebrained at [Boot Process](#)

Firmware Architecture

The HackRF One's LPC4320 dual-core microcontroller has limited resources for doing software-defined radio work. There's a total of 200Kbytes of RAM and 1Mbyte of SPI flash memory.

Division of Labor

There are two cores in the LPC4320:

- Cortex-M4F, which performs baseband signal processing.
- Cortex-M0, which does all user interface tasks, and simple signal/packet post-processing.

The Cortex-M4F firmware has three ChibiOS threads, listed in decreasing priority:

- Baseband: Receives buffers of baseband samples and processes them to recover audio, packets, or whatever.
- RSSI: Receives buffers of RSSI (received signal strength indication) samples and processes them to produce metrics for display. May be used in the future to trigger signal captures and provide receiver AGC for some receiver modes.
- Default: The thread executed inside main() that receives events from the other threads, and messages from the M0 (UI) core.

The Cortex-M0 firmware has only one thread, the default, which waits for events signaled from interrupts and for messages received from the M4 (baseband) core.

Memory Map

Absolute addresses on the HackRF One LPC4320:

```
0x0000_0000 0x1000_0000    Shadow area (controlled by CREG M4MEMMAP and M0APPMEMMAP)
0x1000_0000 0x1001_8000    96k Local SRAM
0x1008_0000 0x1008_8000    32k Local SRAM
0x1008_8000 0x1008_a000     8k Local SRAM
0x1400_0000 0x1410_0000    1M SPIFI flash (cached access to W25Q80BV flash IC)
0x2000_0000 0x2000_8000    32k AHB SRAM
0x2000_8000 0x2000_c000     8k AHB SRAM
0x2000_c000 0x2001_0000     8k AHB SRAM (also ETB)
0x2200_0000 0x2400_0000    32M AHB SRAM bit-banding (only from M4 core?)
0x4000_0000 0x4010_2000     Peripherals
0x4200_0000 0x4400_0000    32M Peripherals bit-banding (only from M4 core?)
0x8000_0000 0x8010_0000    1M SPIFI flash (direct access to W25Q80BV flash IC)
0xe000_0000 0xe010_0000    ARM core private bus (distinct for M4 and M0 cores)
```

How RAM and flash regions are used in PortaPack code:

```
0x1000_0000 0x1001_8000    96k Local SRAM for M4 RAM (stack, heap, data)
0x1008_0000 0x1008_8000    32k Local SRAM for M4 code (text section)
0x1008_8000 0x1008_a000     8k Local SRAM for M4/M0 communication
0x1400_0000 0x1410_0000    1M SPIFI flash for M4 bootstrap, M0 code, M4 code overlays
0x2000_0000 0x2001_0000    64k AHB SRAM for M0 RAM (stack, heap, data)
```

How RAM and flash regions are used in HackRF code (which is launched from PortaPack code):

```
0x1000_0000 0x1001_8000    96k Local SRAM for M4 code
0x1008_0000 0x1008_8000    32k Local SRAM for M4 stack and heap
```

Philosophy

M4 code is run from local SRAM for performance reasons. The smaller local SRAM region is used because DSP code tends to be tight loops and doesn't require much code.

M4 data is kept in a separate local RAM block to maximize performance. The M4 core can retrieve instructions and data from separate blocks simultaneously through use of distinct I-code and D-code buses. The larger local SRAM region is used to maximize storage for baseband data, filter coefficients, and intermediate data.

M0 code is run from flash (mostly) because the UI and non-critical code is much larger than the DSP code on the M4. So the performance hit is outweighed by the extra code storage. The SPIFI interface is cranked up to maximum performance -- 100MHz SCK, quad SPI interface, minimum CS# idle time, fastest SCU mux pins mode, read through cached memory region. Without addressing overhead, this provides 400Mbps transfer, or 50Mbytes/s or 12.5Mwords/sec. Overhead for flash bursts (0xEB instruction) is ~20(!) clocks or 200ns(!). It's unclear how large a region SPIFI caches, but it's probably on the order of 16/64/256 bytes.

M0 data is in the AHB RAM region, to avoid contention with the M4 core's code and data.

There is a small region of VBAT-maintained SRAM (0x40041000, 256 bytes) for saving system state when the device is powered off.

M4 and M0 data RAM is accessed via direct addressing. This permits passing pointers between the M4 and M0. Of course, passing pointers to the M4 or M0 text sections (RAM shadowed/aliased to 0x0) wouldn't work without some sort of address fix-up to point into M4 RAM or the M0 SPIFI image.

TODOs

- Hook up scope and determine SPIFI caching behavior. There may be locality-of-reference or function alignment tricks to improve performance -- when it's needed.
- Investigate using SPI flash 0xEB instruction with M[5:4] = 0b10, to eliminate eight cycles of overhead.

- If SPIFI cache always aligns amenably, perhaps use a different SPI flash instruction that assumes some address LSBs are zero (aligned) to eliminate a cycle or two of overhead.
- If SPIFI transfers always have $A[0] == 0$, use 0xE7 instruction to eliminate two clocks of overhead over 0xEB.
- If SPIFI transfers always have $A[3:0] == 0$, use 0xE3 instruction to eliminate four clocks of overhead over 0xEB.

Original Wiki by sharebrained at [Firmware-Architecture](#)

Persistent Memory

This is part of the **RTC (Real Time Clock) subsystem**, inside the LPC43xx chip on the Portapack hardware.

RTC Subsystem

The RTC subsystem keeps the actual clock (date / time) running, powered by a back up coin battery (which needs to be [installed](#) on the portapack board), even when the unit is not connected to an USB power source. And on top of this RTC functionality, the same coin battery will keep persistent a small region of VBAT-maintained SRAM (0x40041000, 256 bytes) for saving system state settings when the device is powered off.

The persistent memory map is defined at `/firmware/chibios-portapack/os/hal/platforms/LPC43xx/lpc43xx.inc`:

```

**LPC_RTC_DOMAIN_BASE**      (0x40040000)
**LPC_ALARM_TIMER_BASE**     (LPC_RTC_DOMAIN_BASE + 0x0000)
**LPC_BACKUP_REG_BASE**      (LPC_RTC_DOMAIN_BASE + 0x1000) <-backup_ram region!
**LPC_POWER_MODE_CTRL_BASE** (LPC_RTC_DOMAIN_BASE + 0x2000)
**LPC_CREG_BASE**            (LPC_RTC_DOMAIN_BASE + 0x3000)
**LPC_EVENT_ROUTER_BASE**    (LPC_RTC_DOMAIN_BASE + 0x4000)
**LPC_OTP_CTRL_BASE**        (LPC_RTC_DOMAIN_BASE + 0x5000)
**LPC_RTC_BASE**             (LPC_RTC_DOMAIN_BASE + 0x6000)

```

backup_ram region

The **backup_ram region** is a small amount of ram (only 256 bytes), defined in `/firmware/common/memory_map.hpp` using the **LPC_BACKUP_REG_BASE** label, thus starting at address **0x40041000**

data_t struct

The Portapack firmware uses the backup_ram region to store several `uint32_t` (and one `uint64_t`) variables, inside a struct, called "data_t", declared on `/firmware/common/portapack_persistent_memory.cpp`

Variables stored in persistent memory

Found in **original portapack**, but only carrying a few configuration variables, it was later expanded by furrtek's **HAVOC** version, incorporating several new configuration parameters.

Among those new variables, there is a `uint32_t config_t` used as storage for a few boolean flags.

On MAYHEM's version, we've added two new boolean configuration parameters, particularly useful for H1 users including **"Speaker is present"** and **"Back Buttons on menus"**.

You can find them implemented as two new configuration checkboxes inside **OPTIONS->UI**.

These two new boolean parameters have been added into the `uint32_t ui_config` variable, Which now it is mapped as following:

ui_config Bit position / Description

- 31 (HAVOC) Splash screen
- 30 (HAVOC) Login

- 29 (HAVOC)Start in stealth mode
- 28 (**MAYHEM H1 Branch**)Place a back ("..") button in each menu
- 27 (**MAYHEM H1 Branch**)Use speaker output (H1 model accepts a speaker)

Using this two new booleans

They can be reached from anywhere by adding (if not already there) this include:

```
#include "portapack_persistent_memory.hpp"
```

and then you can check the configuration status for each parameter with:

```
portapack::persistent_memory::config_backbutton()
```

and

```
portapack::persistent_memory::config_speaker()
```

They return either TRUE or FALSE, depending on how they've been configured.

ChibiOS Notes

These are miscellaneous useful notes about how ChibiOS functions.

Stacks Configuration

From ChibiOS crt0.c:

Two stacks available for Cortex-M, main stack or process stack.

- **Thread mode:** Used to execute application software. The processor enters Thread mode when it comes out of reset.
- **Handler mode:** Used to handle exceptions. The processor returns to Thread mode when it has finished all exception processing.

ChibiOS configures the Cortex-M in dual-stack mode. (CONTROL[1]=1) When CONTROL[1]=1, PSP is used when the processor is in Thread mode.

MSP is always used when the processor is in Handler mode.

- **main_stack_size:** Used for exception handlers. Yes, really.
- **process_stack_size:** Used by main().

After chSysInit(), the current instructions stream (usually main()) becomes the main thread.

Original Wiki by sharebrained at [Operating System Notes](#)

Code contribution rules

Disclaimer

By contributing you accept to provide code that comply to the following code rules. Any pull request may be stalled until fixed. It's important for everyone, be it contributors or reviewers, to have a clear compile log so they can easily spot and fix new arising problems / warning.

Code contribution rules

- Indent your code. Using automatic indentation is recommended, at least for new apps
- Test and take in account the return code each time it's needed
- Have an error management in your app
- Make reusable content (class, structs, gui filler functions, ...)
- Document global processing as well as tricky parts, don't be lazy, some may contribute to your own contribution. Information inside the code is really appreciated

- Fix your compilation warnings before making a pull request. ALL your warnings. It's making a terrible noise in the compilation log each time we accept PRs with warnings everywhere
- Provide access to your sources via a public repository

A note on Pull Requests

- PRs should be up to date with 'next'
- Each new functionality should have it's own PR, don't put too much files/commits into a single PR. When it's too much commits, one can make a new branch and report only the modified files in a single commit, to make the review easier
- Merging of any PR is up to the maintainers good will

naming and usage rule

- Use English
- `var_and_func_example` (snake_case)
- `_private_member_of_class_example` (_snake_case) (not yet covered all in codebase but new apps. You are welcomed to correct them.)
- `namespaceexample`
- `ClassNameExample` (CamelCase)
- `objectNameExample` (camelCase)
- `func_arg_ex` (snake_case and abbreviation is allowed in small scope functions)
- Don't use `typedef` keyword in C++ code
- `goto` is not allowed

which license of code is safe to use

- MIT: It's safe to copy MIT license, as long as you keep original copyright info.
- MPL: It's safe to copy MPL license, as long as you follow Mozilla's agreement.
- GNU (GPL 2/3 / AGPL): This project is under GPL which means it's safe to use GPL code here as long as you open source your the distro under GPL license.
- The Unlicensed: safe to use.
- BSD: safe to use.
- WTFPL: Safe to use.
- CC0: safe to use.

Create a Simple App

About

This wiki will go over how to create a simple application and show you the basics in the Mayhem code base. This is to be considered a "hello world" application. :)

We are going to create few simple controls and link the app to the main menu. The idea is to get familiar with the way things work.

First steps

There is a lot of flexibility, but for now we are going to follow patterns already found on the source code. For example, the name of the files and classes will be inspired by existing code.

Application code

The following structure is the base of any application. Following the general structure, this files in this example will be created on `firmware\application\apps\`

ui_newapp.hpp

```
#include "ui.hpp"
#include "ui_widget.hpp"
#include "ui_navigation.hpp"
#include "string_format.hpp"

namespace ui
{
    class NewAppView : public View // App class declaration
    {
    public:
        NewAppView(NavigationView &nav); // App class init function declaration
        std::string title() const override { return "New App"; }; // App title

    private:
        void update(); // Function declaration
        MessageHandlerRegistration message_handler_update{ // Example, not required: MessageHandler
            Message::ID::DisplayFrameSync, // relays messages to your app code from
            [this](const Message *const) { // get a DisplayFrameSync message the u
                this->update(); // be triggered.
            }
        };
    };
}
```

ui_newapp.cpp

```
#include "ui_newapp.hpp"
#include "portapack.hpp"
#include <cstring>

using namespace portapack;

namespace ui
{
    NewAppView::NewAppView(NavigationView &nav) // Application Main
    {
        // App code
    }

    void NewAppView::update() // Every time you get a DisplayFrameSync message this func
    {
        // Message code
    }
}
```

Entry in the main menu

For triggering your new app, you need to add an entry on the main menu. This menu resides on [firmware\application\ui_navigation.cpp](#). Check the current entries, and add a new one in a section you think is suitable for your new app.

firmware\application\ui_navigation.cpp

```
// Add this to the top to link your new app's header file
#include "ui_newapp.hpp"

...

// Adding NewApp to the Transmitters Menu
const NavigationView::AppList NavigationView::appList = {
    add_items{
```

```

...
{"newapp", "NewApp", TX, Color::red(), &bitmap_icon_remote, new ViewFactory<NewAppView>()},
});
}

```

Early test

Remember to add your apps/ui_newapp.cpp to firmware\application\CMakeLists.txt

firmware\application\CMakeLists.txt

```

# C++ sources that can be compiled in ARM or THUMB mode depending on the global
# setting.
set(CPPSRC
    main.cpp

    ...

    apps/ui_newapp.cpp
)

```

In this moment you should be able to compile and test the app in your device. For your reference here's the link to the [Compile-firmware](#) wiki. The new app should appear in the menu you picked in ui_navigation.cpp.

Adding Widgets and Basic Functionality

Widgets are the elements that compose the UI of your custom app. Widgets are defined inside [firmware\common\ui_widget.hpp](#) and widget functions can be found inside [firmware\common\ui_widget.cpp](#). In order to be able to use them, you must `#include "ui_widget.hpp"` into your app .hpp file. There are different type of widget. here you will find a list of available widget, with their respecting constructor. For all the methods available, you should go and see the [ui_widget.hpp](#) file.

Attach a Generic Widget to Your Application

In order to display a widget into your app, you can either use the function `add_child()` or `add_children()`. Both those functions shall be called within the code of your `NewAppGameView(NavigationView &nav){}` constructor. The difference between the two function is simple: the first one allows you to add a single widget, while the second one allows you to add an undefined number of widgets. Widgets must be passed as pointers to the functions. A correct way of calling the two functions would then be:

```
add_child(&my_widget);
```

or

```
add_children({
    &widget_1,
    &widget_2
});
```

There are several different widgets that we're going to use for our new app. A more complete list can be found on the [Widgets](#) wiki. More might be added so you should always go and check whether new widgets have been added or not.

Button

Buttons allows you to do something when you press them. Here you can find it's declaration and prototype:

```
Button my_button_widget{
    Rect parent_rect,
    std::string text
};
```

Be aware that every time you create a button, you then have to implement this method:

`my_button_widget.on_select = [&nav](Button &){}`. You can leave it empty (even though it should not, as here you define what action the button should perform), but it must be present in your code.

For example, let's say you want a button called `my_button`, with the same dimensions as the previous widget. You will then do:

```
Button my_button(
    {10, 10, 100, 24}, // Coordinates are: int:x (px), int:y (px), int:width (px), int:height (px)
    "my_button_text"
);
```

Labels

Labels are a text element that can be used to describe other widgets. Here you can find it's declaration and prototype:

```
Labels my_label_widget{
    std::initializer_list<Label> labels
};
```

For example, let's say you want a label called `my_label`. Because the constructor is looking for list you'll need to add a set of brackets `{}` around each label. You will need to add this to `apps/ui_newapp.hpp`:

```
Labels my_label{
    {{10, 10},           // Coordinates are: int:x (px), int:y (px)
    "my_label_text:",    // Label text
    Color::light_grey(), // Label color
};
```

Note: Colors are defined in [firmware/common/ui.hpp](#).

In `apps/ui_newapp.cpp` you'll need to add the `my_label` pointer to `add_child()` or `add_children()`:

```
NewAppView::NewAppView(NavigationView &nav) {
    // Widget pointers
    add_children({
        &my_label,
    });
}
```

LiveDateTime

LiveDateTime gives you the dynamic date and time. Here you can find it's declaration and prototype:

```
LiveDateTime my_liveDateTime_widget{
    Rect parent_rect
```

```
};
```

For example, let's say you want a label called `my_liveDateTime`. You will need to add this to `apps/ui_newapp.hpp`:

```
LiveDateTime my_liveDateTime {  
    { 2, 10, 19*8, 16 },          // Coordinates are: int:x (px), int:y (px), int:width (px), int:height (px)  
};
```

In `apps/ui_newapp.cpp` you'll need to add the `my_liveDateTime` pointer to `add_child()` or `add_children()`:

```
NewAppView::NewAppView(NavigationView &nav) {  
  
    // Widget pointers  
    add_children({  
        &my_liveDateTime,  
    });  
  
}
```

If you want to enable seconds you'll need use the `set_seconds_enabled(bool new_value)` function:

```
my_liveDateTime.set_seconds_enabled(true);
```

ProgressBar

Progress bars are a visual representation of progress that let us know how far a long a task is. Here you can find it's declaration and prototype:

```
Labels my_progressBar_widget{  
    Rect parent_rect  
};
```

For example, let's say you want a label called `my_progressBar`. You will need to add this to `apps/ui_newapp.hpp`:

```
ProgressBar my_progressBar {  
    { 2, 10, 208, 16 },          // Coordinates are: int:x (px), int:y (px), int:width (px), int:height (px)  
};
```

In `apps/ui_newapp.cpp` you'll need to add the `my_progressBar` pointer to `add_child()` or `add_children()`:

```
NewAppView::NewAppView(NavigationView &nav) {  
  
    // Widget pointers  
    add_children({  
        &my_progressBar,  
    });  
  
}
```

To set the maximum value for the progress bar use the `set_max(const uint32_t max)` function:


```
my_progressBar.set_max(10); // 10 is 100%
```

To change the value of progress for the progress bar use the `set_value(const uint32_t value)` function:

```
my_progressBar.set_value(5); // 50% Complete
```

NumberField

NumberField is similar to the OptionsField widget except that it only deals with numbers. You can change its value with the wheel on your portapack. Here you can find its declaration and prototype:

```
NumberField my_NumberField_widget{
    Point parent_pos,
    int length,
    range_t range,
    int32_t step,
    char fill_char,
    bool can_loop
};
```

For example, let's say you want a NumberField called `my_numberField`. You will need to add this to `apps/ui_newapp.hpp`:

```
// Example 3 digit number starting at "000", ends at "255"
NumberField my_numberField(
    {10, 10},           // Coordinates are: int:x (px), int:y (px)
    3,                  // Length
    {0, 255},           // MIN -> MAX Range
    1,                  // Step
    '0',                // Fill Char
    false               // Can Loop
);
```

In `apps/ui_newapp.cpp` you'll need to add the `my_numberField` pointer to `add_child()` or `add_children()`:

```
NewAppView::NewAppView(NavigationView &nav) {
    // Widget pointers
    add_children({
        &my_numberField,
    });
}
```

Functions within `apps/ui_newapp.cpp` are able to lookup the value of `my_numberField` with NumberField's `value()` function:

```
int number = my_numberField.value();
```

You can also set the value for `my_numberField` with the `set_value(int32_t new_value, bool trigger_change)` function:

```
my_numberField.set_value(123);
```

If you want your NumberField to change a value (int number for example) you'll need to add this [Lambda](#) to apps/ui_newapp.cpp:

```
NewAppView::NewAppView(NavigationView &nav) {  
    // Add widget pointers  
    add_children({  
        &my_numberField,  
    });  
  
    // When NumberField is changed  
    my_numberField.on_change = [this](int32_t v) {  
        number = v;  
    };  
}
```

Wrap Up

Bellow is an example "Hello World" application that shows off a few widgets and logic that controls their functions.

ui_newapp.hpp

```
#include "ui.hpp"  
#include "ui_widget.hpp"  
#include "ui_navigation.hpp"  
#include "string_format.hpp"  
  
// Define a constant  
#define PROGRESS_MAX 100  
  
namespace ui  
{  
    class NewAppView : public View // App class declaration  
    {  
    public:  
  
        // Public declarations  
        void focus() override; // ui::View function override  
  
        NewAppView(NavigationView &nav); // App class init function declaration  
        std::string title() const override { // App Title  
            return "New App";  
        };  
  
    private:  
  
        // Private declarations  
        void update(); // Function declaration  
        MessageHandlerRegistration message_handler_update{ // Example, not required: MessageHandlerRegistr  
            Message::ID::DisplayFrameSync, // relays machine states to your app code. Eve  
            [this](const Message *const) { // get a DisplayFrameSync message the update  
                this->update(); // be triggered.  
            }  
        };  
  
        // Variables  
        uint32_t progress = 0;  
  
        // Widgets  
        // Note: Usable screen space is 240x304px  
        // Note: Each char takes up 8x8px so you can multiply  
        // the amount of spaces and rows you want by 8.  
        // This gives you 30x38 char  
  
        Button button_helloWorld{  
            {70, 128, 100, 24}, // Coordinates are: int:x (px), int:y (px), int:width (px), int  
            "Hello World!" // Title  
        }  
    }  
}
```

```

};

LiveDateTime timestamp {
    {6*8, 22*8, 19*8, 20 }           // Coordinates and Dimensions
};

Labels label_progress {
    {{8*8, 33*8},                     // Coordinates are: int:x(px), int:y(px)
    "Progress:   %",                   // Title
    Color::light_grey()}               // Title color
};

NumberField numberField_progress {
    {18*8, 33*8},                     // Coordinates
    3,                                 // Length of number
    {0, PROGRESS_MAX},                 // Range
    1,                                 // Step
    '0',                               // Fill Char
    false                             // Loop?
};

ProgressBar progressBar_progress {
    {2*8, 35*8, 208, 16 },            // Coordinates and Dimensions
};

};

```

ui_newapp.cpp

```

#include "ui_newapp.hpp"
#include "portapack.hpp"
#include <cstring>

using namespace portapack;

namespace ui
{
    void NewAppView::focus() {           // Default selection to button_helloWorld when app starts
        button_helloWorld.focus();
    }

    NewAppView::NewAppView(NavigationView &nav) // Application Main
    {
        add_children({                     // Add pointers for widgets
            &button_helloWorld,
            &label_progress,
            &numberField_progress,
            &progressBar_progress,
            &timestamp,
        });

        progressBar_progress.set_max(PROGRESS_MAX); // Set max for progress bar

        button_helloWorld.on_select = [this](Button &){ // Button logic
            if(progress < 100) {
                numberField_progress.set_value(100); // Because numberField_progress has an on_change event
            } else { // progressBar_progress will update automatically
                numberField_progress.set_value(0);
            }
        };

        numberField_progress.on_change = [this](int32_t v) { // When NumberField is changed
            progress = v;
            progressBar_progress.set_value(progress);
        };

        timestamp.set_seconds_enabled(true); // DateTime enable seconds
    }
}

```

```
void NewAppView::update()
{
    // Message code
}
```

```
// Every time you get a DisplayFrameSync mess
// function will be ran.
```

Building External Apps

External apps are similar to internal apps in that they are compiled and linked with the rest of the firmware, but at the link stage a fake memory address range is specified for each external app in the "external.ld" file. After linking, the external apps are removed from the firmware image and extracted to separate files. This is quite different from baseband modules that are each compiled and linked separately, run on a different CPU (the M4), and communicate with the main firmware via a message protocol. Note that LTO optimization cannot be used when linking the external apps, to prevent the linker from attempting to share code between external apps.

export_external_apps.py

The `export_external_app` python script edits each external app image to (1) replace the fake memory addresses used during the linker stage with actual RAM addresses where the external app will be loaded to be executed, (2) append any needed baseband image, and (3) adds a file checksum to verify integrity during the untar process.

External App Address Replacement

As example, if the fake address range of an app is `0xADC00000` to `0xADC07FFF`, this python script will search the external app file for values in this range and replace them; it *assumes* that they are pointers to memory addresses within this app. If the app image contains a value where the high byte is `0xAD` but the next most significant byte is potentially an address within a different app, a warning message is displayed such as the following. This message implies that code within the indicated app may be attempting to use code within another app (but since the LTO optimization is disabled, it is most likely that this warning message is a false positive and the value found could just be a code instruction or raw data):

WARNING: External code address 0xad01234 at offset 0x1000 in tetris.himg

Note that the fake `0xADxxxxxx` address range was selected for external apps based on few data values in this range in the firmware image, and because any firmware attempts to access this fake memory address range will trigger a GURU meditation fault.

External App Baseband Images

Baseband modules for external apps are created exactly the same as baseband images for internal apps, are compressed, and execute from RAM on the M4 processor as all baseband modules do. But, external app baseband images are omitted from the firmware ROM and are instead appended to the external app image using the python script mentioned above.

External App Checksum

The checksum used for external app image files is simply a `uint32` sum of all 32-bit values in the image file, with the sign finally inverted such that the total (including checksum) will sum to 0 if it's a valid image. This simple checksum method is used for speed (CRC32 would be slower).

make_spi_image.py

The `make_spi_image` python script creates the firmware ROM image by appending the baseband images and a simple checksum, and also checks to make sure there are no references to the "fake" memory address regions used for external apps mentioned above. The checksum algorithm is the same as used for external apps, above. A warning message similar to the one below may be displayed if data values are found in the firmware image that *might* be an attempt to access a "fake" memory address region:

WARNING: Possible external code address 0xadb96ef0 at offset 0xb24a4 in portapack-h1_h2-mayhem.bin

To determine if a warning message such as that above is real or a false positive, search for the mentioned external code address within the memory regions indicated in the external.ld file. In this example, you might find a line in external.ld like this, which implies that firmware might possibly be trying to access memory in the LCR app:

```
ram_external_app_lcr(rwx) : org = 0xADB90000, len = 32k
```

To determine if it's real or a false positive, edit this line of the external.ld file to change the LCR app's address to another range that is unused such as "org = 0xADFF0000" and rebuild firmware. If the address in the warning message changes to the new address range, then firmware really is trying to access code or data within that external app. If the address in the warning message stays the same, then it's a false positive and can safely be ignored. (Future updates to the python script will hopefully eliminate the bogus warnings.)

Widgets

About Widgets

Widgets are elements to compose the UI of your custom app. Widgets are defined in [firmware\common\ui_widget.hpp](#) and widget functions can be found in [firmware\common\ui_widget.cpp](#). To use them, you must add the line `#include "ui_widget.hpp"` into your app .hpp file.

There are different types of widgets. Here you will find a list of available widgets, with their constructor. For all methods available, you should check the [ui_widget.hpp](#) file.

Attach a generic Widget to your Application

To display a widget in your app, you can use the functions `add_child()` for a single widget or `add_children()` to add an undefined number of widgets.

Both functions shall be called within the code of the `NewAppGameView(NavigationView &nav){}` constructor.

Widgets must be passed as pointers to the functions.

Declaration and prototype

```
add_child(&my_widget);
```

or

```
add_children({
    &widget_1,
    &widget_2
});
```

Available widgets

There are several widgets available and more might be added. You should always check [ui_widget.hpp](#) if new widgets have been added.

In this document you will find a list of most basic widgets.

Text

The text widget add a simple text area widget in the app.

Declaration and prototype

```
Text my_text_widget{
    Rect parent_rect,
    std::string text
};
```

Note: Rect parent_rect has it's own definition inside another file.

Example

Add a text widget with the content "Hello World" positioned at 10x10 px from the top left corner with width 100 and height 24.

```
Text hello_world_text_widget(
    {10, 10, 100, 24}, // Coordinates are: int:x (px), int:y (px), int:width (px), int:height (px)
    "Hello world!"
);
```

Button

Buttons allows to do something when you press it.

Declaration and prototype

```
Button my_button_widget{
    Rect parent_rect,
    std::string text
};
```

Note: Every time you create a button, you have to implement the method `my_button_widget.on_select = [&nav](Button &){}`. It could be empty (even though it shouldn't, as here you define the action for a button), but it must be present in the code.

Example

A button called `my_button`, with the same dimensions as the previous widget.

```
Button my_button(
    {10, 10, 100, 24}, // Coordinates are: int:x (px), int:y (px), int:width (px), int:height (px)
    "my_button_text"
);
```

Labels

Labels are a text element that can be used to describe other widgets.

Declaration and prototype

```
Labels my_label_widget{
    std::initializer_list<Label> labels
};
```

Example

A label called `my_label1`. Because the constructor is looking for a list you'll need to add brackets `{ }` around each label.

Add this to `apps/ui_newapp.hpp`:

```
Labels my_label{
    {{10, 10}, // Coordinates are: int:x (px), int:y (px)
    "my_label_text:", // Label text
```

```
Color::light_grey()), // Label color
};
```

Note: Colors are defined in [firmware/common/ui.hpp](#).

In `apps/ui_newapp.cpp` add the `my_label` pointer to `add_child()` or `add_children()`:

```
NewAppView::NewAppView(NavigationView &nav) {
    // Widget pointers
    add_children({
        &my_label,
    });
}
```

LiveDateTime

`LiveDateTime` gives the dynamic date and time.

Declaration and prototype

```
LiveDateTime my_liveDateTime_widget{
    Rect parent_rect
};
```

Example

For a label called `my_liveDateTime` add the following code to `apps/ui_newapp.hpp`:

```
LiveDateTime my_liveDateTime {
    { 2, 10, 19*8, 16 }, // Coordinates are: int:x (px), int:y (px), int:width (px), int:height (px)
};
```

In `apps/ui_newapp.cpp` add the `my_liveDateTime` pointer to `add_child()` or `add_children()`:

```
NewAppView::NewAppView(NavigationView &nav) {
    // Widget pointers
    add_children({
        &my_liveDateTime,
    });
}
```

To enable seconds use the function `set_seconds_enabled(bool new_value)`:

```
my_liveDateTime.set_seconds_enabled(true);
```

BigFrequency

`BigFrequency` is used for displaying a radio frequency.

Declaration and prototype

```
BigFrequency my_bigFrequency_widget{
    Rect parent_rect,
```

```
rf::Frequency frequency
};
```

Example

For a big frequency widget called `my_bigFrequency` add the following code to `apps/ui_newapp.hpp`:

```
BigFrequency my_bigFrequency(
    {10, 10, 28*8, 52}, // Coordinates are: int:x (px), int:y (px), int:width (px), int:height (px)
    0 // Beginning frequency in hz
);
```

In `apps/ui_newapp.cpp` add the `my_bigFrequency` pointer to `add_child()` or `add_children()`:

```
NewAppView::NewAppView(NavigationView &nav) {

    // Widget pointers
    add_children({
        &my_bigFrequency,
    });

}
```

To set a frequency use the function `set(const rf::Frequency frequency)`:

```
my_bigFrequency.set(433000000); // 433MHz
```

ProgressBar

Progress bars are a visual representation of the progress of a task.

Declaration and prototype

```
Labels my_progressBar_widget{
    Rect parent_rect{https://github.com/portapack-mayhem/mayhem-firmware.wiki.git}
};
```

Example

For a label called `my_progressBar` add the following code to `apps/ui_newapp.hpp`:

```
ProgressBar my_progressBar {
    { 2, 10, 208, 16 }, // Coordinates are: int:x (px), int:y (px), int:width (px), int:height (px)
};
```

In `apps/ui_newapp.cpp` add the `my_progressBar` pointer to `add_child()` or `add_children()`:

```
NewAppView::NewAppView(NavigationView &nav) {

    // Widget pointers
    add_children({
        &my_progressBar,
    });

}
```

Set the maximum value for the progress bar with the function `set_max(const uint32_t max)`:

```
my_progressBar.set_max(10); // 10 is 100%
```


Change the value of progress for the progress bar with the function `set_value(const uint32_t value)`:

```
my_progressBar.set_value(5); // 50% Complete
```

Console

Console can be used as large text field where you can output mutable lines of text.

Declaration and prototype

```
Console my_console_widget{  
    Rect parent_rect  
};
```

Example

For a label called `my_console` add the following code to `apps/ui_newapp.hpp`:

```
Console my_console {  
    { 2*8, 10, 208, 200 },    // Coordinates are: int:x (px), int:y (px), int:width (px), int:height (px)  
};
```

In `apps/ui_newapp.cpp` add the `my_console` pointer to `add_child()` or `add_children()`:

```
NewAppView::NewAppView(NavigationView &nav) {  
  
    // Widget pointers  
    add_children({  
        &my_console,  
    });  
}
```

To write to 'my_console' use the function `write(std::string message)`:

```
my_console.write("Hello World");
```

For automatic new line use the function `writeln(std::string message)`. It will add `\n` at the end of every string:

```
my_console.writeln("Hello World but on a new line!");
```

To enable scrolling use the function `enable_scrolling(bool enable)`:

```
my_console.enable_scrolling(true);
```

Note: The buffer size is limited to 256 char

Checkbox

Checkbox is a boolean (true/false) widget, that allows you to chose between two options. The Checkbox is displayed as a rectangle with a green check mark (true) or a red cross (false).

Declaration and prototype

```
Checkbox my_checkbox_widget{
    Point parent_pos,
    size_t length,
    std::string text,
    bool small
};
```

Example

For a checkbox called `my_checkbox` add the following code to `apps/ui_newapp.hpp`:

```
Checkbox my_checkbox(
    {10, 20},           // Coordinates are: int:x (px), int:y (px)
    4,                  // Length
    "my_checkbox_text", // Title
    false               // Checkbox Size: true == small(16X16px), false == regular(24X24px)
);
```

In `apps/ui_newapp.cpp` add the `my_checkbox` pointer to `add_child()` or `add_children()`:

```
NewAppView::NewAppView(NavigationView &nav) {

    // Widget pointers
    add_children({
        &my_checkbox,
    });

}
```

Functions within `apps/ui_newapp.cpp` can lookup the value of `my_checkbox` with `Checkbox's value()` function:

```
if(my_checkbox.value()) {
    do_a();                // If checkbox is selected (green check mark)
} else {
    do_b();                // If checkbox is NOT selected (red X)
}
```

Set the value for `my_checkbox` with the function `set_value(const bool value)`:

```
my_checkbox.set_value(true); // Checkbox selected (green check mark)
my_checkbox.set_value(false); // Checkbox deselected (red X)
```

To automatically perform an action when the checkbox is toggled, add [Lambda](#) to `apps/ui_newapp.cpp`:

```
NewAppView::NewAppView(NavigationView &nav) {

    // Add widget pointers
    add_children({
        &my_checkbox,
    });

    // When checkbox is toggled do...
    my_checkbox.on_select = [this](Checkbox&, bool v) {
        if(v){
            do_a();                // If checkbox is selected (green check mark)
        } else {
            do_b();                // If checkbox is NOT selected (red X)
        }
    };

}
```

Image

Images can be displayed within your app.

Declaration and prototype

```
Image my_Image_widget{
    Rect parent_rect,
    Bitmap* bitmap,
    Color foreground,
    Color background
};
```

Images need to be a Bitmap object before they can be displayed. Below is an example of the code needed to create a Bitmap from [firmware/application/bitmap.hpp](#).

```
static constexpr uint8_t bitmap_stripes_data[] = {
    0xFF, 0x03, 0xC0,
    0xFF, 0x01, 0xE0,
    0xFF, 0x00, 0xF0,
    0x7F, 0x00, 0xF8,
    0x3F, 0x00, 0xFC,
    0x1F, 0x00, 0xFE,
    0x0F, 0x00, 0xFF,
    0x07, 0x80, 0xFF,
};
static constexpr Bitmap bitmap_stripes {
    { 24, 8 }, bitmap_stripes_data
};
```

Example

With the Bitmap object created we can define the image `my_image`. Add the following code to `apps/ui_newapp.hpp`:

```
Image my_image(
    {10, 10, 24, 8},    // Coordinates are: int:x (px), int:y (px), int:width (px), int:height (px)
    &Bitmap,             // Pointer to your bitmap
    Color::white(),      // Color Foreground
    Color::black()       // Color Background
);
```

Note: Colors are defined in [firmware/common/ui.hpp](#)

In `apps/ui_newapp.cpp` add the `my_image` pointer to `add_child()` or `add_children()`:

```
NewAppView::NewAppView(NavigationView &nav) {
    // Widget pointers
    add_children({
        &my_image,
    });
}
```

OptionsField

OptionsField is a widget to create a field, in which you can change its value with the wheel on your PortaPack.

Declaration and prototype

```
OptionsField my_OptionsField_widget{
    Point parent_pos,
    int length,
    options_t options,
    bool centered,
};
```

parent_pos is an array of two integer, represents where the top left corner of the widget should be positioned. The **length** is an integer which tells how many options you have into your options parameter. The **options_t** field is an array of options in which your portapack can choose to display.

Centered is a bool that if you true it, you can make all the text in it in the center. It's originally init-ed, so if you don't define it, it will be default false.

Example

An OptionsField called **my_optionsField**, with 3 options positioned at 10 from top and 10 from left:

```
OptionsField my_optionsField{
    {10,10},           // Coordinates are: int:x (px), int:y (px)
    7,                 // Char length for option title
    {
        {"option1",0},
        {"option2",1},           // Options {"KEY", int VALUE}
        {"option3",2}
    }
};
```

Note: The number following the **option_n** string value, should be the value that could be retrieved from the OptionsField with the function **my_optionsField.selected_index_value()**;

NumberField

NumberField is similar to the OptionsField widget except that it only deals with numbers. You can change its value with the wheel on your portapack.

Declaration and prototype

```
NumberField my_NumberField_widget{
    Point parent_pos,
    int length,
    range_t range,
    int32_t step,
    char fill_char,
    bool can_loop
};
```

Example

For a NumberField called **my_numberField** add the following code to **apps/ui_newapp.hpp**:

```
// Example 3 digit number starting at "000", ends at "255"
NumberField my_numberField(
    {10, 10},           // Coordinates are: int:x (px), int:y (px)
    3,                 // Length
    {0, 255},           // MIN -> MAX Range
    1,                 // Step
    '0',               // Fill Char
    false              // Can Loop
);
```

In **apps/ui_newapp.cpp** add the **my_numberField** pointer to **add_child()** or **add_children()**:

```
NewAppView::NewAppView(NavigationView &nav) {

    // Widget pointers
    add_children({
        &my_numberField,
    });

}
```

Functions within `apps/ui_newapp.cpp` are able to lookup the value of `my_numberField` with `NumberField's value()` function:

```
int number = my_numberField.value();
```

Set the value for `my_numberField` with the function `set_value(int32_t new_value, bool trigger_change)`:

```
my_numberField.set_value(123);
```

If you want your `NumberField` to change a value (int number for example) you'll need to add [Lambda](#) to `apps/ui_newapp.cpp`:

```
NewAppView::NewAppView(NavigationView &nav) {

    // Add widget pointers
    add_children({
        &my_numberField,
    });

    // When NumberField is changed
    my_numberField.on_change = [this](int32_t v) {
        number = v;
    };

}
```

Waveform

Waveforms are used to display a sign wave from a signal source.

Note: The X axis represents time while the Y axis represents amplitude.

Declaration and prototype

```
Labels my_waveform_widget{
    Rect parent_rect,
    int16_t * data,
    uint32_t length,
    int32_t offset,
    bool digital,
    Color color
};
```

Example

For a waveform widget called `my_waveform` add the following code to `apps/ui_newapp.hpp`:

```
Waveform my_waveform(
    {0, 5*16, 240, 64}, // Coordinates are: int:x (px), int:y (px), int:width (px), int:height (px)
    waveform_buffer,    // RX data
    128,                // Length, how many elements in the waveform_buffer array
    0,                  // Offset
```

```

    false,                // Digital
    Color::white()        // Sine wave color
);

```

Note: Colors are defined in [firmware/common/ui.hpp](#).

The data being displayed by `my_waveform` needs to be a `int16_t` array. Declare this variable in `apps/ui_newapp.hpp`:

```

class NewAppView : public View
{
    public:

        ...

    private:
        int16_t waveform_buffer[128]; // Data for Waveform

        ...
};

```

In `apps/ui_newapp.cpp` add the `my_waveform` pointer to `add_child()` or `add_children()`:

```

NewAppView::NewAppView(NavigationView &nav) {

    // Widget pointers
    add_children({
        &my_waveform,
    });

}

```

If the input data has a variable length, use the function `set_length(const uint32_t new_length)` to update the waveform:

```

my_waveform.set_length(9001) // THAT'S OVER 18KB!!

```

VuMeter

VuMeter is used to visually represent the sound intensity of an audio source.

Declaration and prototype

```

Labels my_vuMeter_widget{
    Rect parent_rect,
    uint32_t LEDs,
    bool show_max
};

```

Example

For a VuMeter called `my_vuMeter` add the following code to `apps/ui_newapp.hpp`:

```

VuMeter my_vuMeter(
    { 0*8, 1*8, 2*8, 33*8}, // Coordinates are: int:x (px), int:y (px), int:width (px), int:height (px)
    12,                      // LEDs
    true                     // Show max
);

```

In `apps/ui_newapp.cpp` add the `my_vuMeter` pointer to `add_child()` or `add_children()`:

```
NewAppView::NewAppView(NavigationView &nav) {

    // Widget pointers
    add_children({
        &my_vuMeter,
    });

}
```

Set the value for my_vuMeter with the function `set_value(const uint32_t new_value):`

```
my_vuMeter.set_value(123); // Max is 255
```

SD Card

About

Reading and writing to the SD card is a great way for your application to save progress or reference data that might be too large to fit in the firmware. Below are examples on how to interface with the SD card using the Mayhem code base.

File Class

Most of the heavy lifting for working with files on the SD Card is done by the [File](#) class. This helper class which simplifies the `FatFs - Generic FAT file system module` which can be found at [firmware/chibios-portapack/ext/fatfs/src/ff.c](#). This wiki will go over some examples on how to read and write files to the SD card.

Continuing with the [Create a Simple App](#) the code below will outline what is required to manipulate the file system. The first thing you'll need to do is include the File class and SD Card helper functions to your application's hpp file.

ui_newapp.hpp

```
// Add these to include File Class and SD Card helper functions.
#include "file.hpp"
#include "sd_card.hpp"
```

Check SD Card

Before reading and writing from the SD Card it's ideal to do a quick check to see if the SD card is mounted. There is error handling working in the background that keeps things from crashing however this will give people a better idea if there's a problem or not. The function below does a quick check to see if the SD Card is showing a status of "Mounted". This will return true if `sd_card::status()` returns "Mounted" or false if `sd_card::status()` returns any other status. SD Card statuses are defined in [firmware/application/sd_card.hpp](#).

ui_newapp.cpp

```
// Checks SD Card, returns true if Mounted, false if otherwise
bool NewAppView::check_sd_card() {
    return (sd_card::status() == sd_card::Status::Mounted) ? true : false;
}
```

Below is a quick example on how this error check could be used.

ui_newapp.cpp

```
// Check SD Card
if(check_sd_card()) {
    // Logic if SD Card is mounted
} else {
    // Logic if SD Card is NOT mounted
}

// Check to see if SD Card is mounted
// Else, check_sd_card() returned false
```

List Directory Contents

To make things easier to use we're going to use a method from [firmware/application/apps/ui_fileman.cpp](#) to list the contents of the SD Card. First thing we'll need to do is create a Struct that outlines the pertinent information of files and directories. Each struct will include the file path, size, and if the item is a directory or not.

ui_newapp.hpp

```
// Struct that outlines file information
struct file_entry {
    std::filesystem::path entry_path { };
    uint32_t size { };
    bool is_directory { };
};
```

Below is an function that will return a [Vector](#) (simmer to a list) of `file_entry` Struts. The input for this function needs a `std::filesystem::path` which can be UTF-16 string literal. The root of the SD Card is `u""` and any directory beyond that is `u"DIRECTORY/SUB_DIRECTORY"`.

This function will also place any directories at the front of the Vector and files in the back. This will also skip any files that are hidden as in anything with a `.` in front of the file name.

ui_newapp.cpp

```
// Lists all files and directories in path
std::vector<file_entry> NewAppView::list_dir(const std::filesystem::path& path) {

    // Files and directories list
    std::vector<file_entry> entry_list { };

    // For each entry in the file system's directory
    // Adds files in directories into entry_list{}
    // Directories are inserted in front of files
    for (const auto& entry : std::filesystem::directory_iterator(path, u"")) {

        // Dose not add directorys or files starting with '.' (hidden / tmp)
        if (entry.path().string().length() && entry.path().filename().string()[0] != '.') {

            // If file
            if (std::filesystem::is_regular_file(entry.status())) {
                entry_list.push_back({ entry.path(), (uint32_t)entry.size(), false });
            }

            // Else If directory
            } else if (std::filesystem::is_directory(entry.status())) {
                entry_list.insert(entry_list.begin(), { entry.path(), 0, true });
            }

            // Other
            } else {
                continue;
            }
        }
    }
}
```



```
    return entry_list;
}
```

Create Directory

To create a directory we can use the `make_new_directory()` function from [File](#). The helper function below will return true if `make_new_directory()` succeeds (Success from `make_new_directory()` returns a 0, other values means failure) and takes two variable inputs for file path and directory name. Again the input for the file path is a `std::filesystem::path` which can be UTF-16 string literal. The root of the SD Card is `u""` and any directory beyond that is `u"DIRECTORY/SUB_DIRECTORY"`.

ui_newapp.cpp

```
// Creates dir, returns true if successful
bool NewAppView::create_dir(const std::filesystem::path& path, std::string name) { // make_new_directory(
    return !(make_new_directory(path.string() + "/" + name));                      // and other values me
}                                                                                   // will return true or
```

Below is an example on how the `create_dir()` function can be used with basic error handling.

ui_newapp.cpp

```
// New directory
if(check_sd_card()) { // Check to see if SD Card is mounted
    if(create_dir(u"", "NEW_DIR")) { // New dir in root of SD Card, returns true if successful
        // Logic if new dir succeeded
    } else {
        // Logic else new dir failed
    }
} else { // Else, check_sd_card() returned false
    // Logic else SD Card is NOT mounted
}
```

Create File

To create a file we can use the `create()` function from [File](#) class. The helper function below will return true if `create()` succeeds (Success from `create()` returns a 0, other values means failure) and takes two variable inputs for file path and directory name. Again the input for the file path is a `std::filesystem::path` which can be UTF-16 string literal. The root of the SD Card is `u""` and any directory beyond that is `u"DIRECTORY/SUB_DIRECTORY"`.

ui_newapp.cpp

```
bool NewAppView::create_file(const std::filesystem::path& path, std::string name) {
    File file = { }; // Create File object
    Optional<File::Error> sucess = file.create(path.string() + "/" + name); // Create File
    return !(sucess.is_valid()); // 0 is success
}
```

Below is an example on how the `create_file()` function can be used with basic error handling.

ui_newapp.cpp

```
// New file
if(check_sd_card()) { // Check to see if SD Card is mounted
    if(create_file(u"", "NEW_FILE.txt")) { // New file in root of SD Card, returns true if successful
```

```

        // Logic if new file succeeded
    } else {
        // Logic else new file failed
    }
} else {
    // Else, check_sd_card() returned false
    // Logic else SD Card is NOT mounted
}

```

A word about reading and writing

In both mode a hardware limit is preventing any buffered read higher than blocks of 512 to be read or write

!! While you can and will do buffered read/write, the size of the buffer must not be higher than 512 !!

See the [related issue discussion](#)

Read File

To read to a file the `read_file()` function from the [File](#) class can be used. The helper function below will return true if the class object `File` opens the file successfully. Success from this open function will have a `is_valid()` function which will return a 0, other values means failure. Inputs for file path and directory name. Again the input for the file path is a `std::filesystem::path` which can be UTF-16 string literal. The root of the SD Card is `u""` and any directory beyond that is `u"DIRECTORY/SUB_DIRECTORY"`

Note: The below sample code dose not handle large files, memory management needs to be implemented.

ui_newapp.cpp

```

std::string NewAppView::read_file(const std::filesystem::path& path, std::string name) { // Read file
    std::string return_string = ""; // String to be r
    File file; // Create File ob
    auto success = file.open(path.string() + "/" + name); // Open file to v

    if(!success.is_valid()) { // 0 is success
        char one_char[1]; // Read file char
        for(size_t pointer = 0; pointer < file.size() ; pointer++) { // Example won't
            file.seek(pointer); // Sets file to r
            file.read(one_char, 1); // sets char to c
            return_string += one_char[0]; // Add it to the
        }
    } else {
        return "0"; // Basic error ha
    }
    return return_string;
}

```

ui_newapp.cpp

```

if(check_sd_card()) { // Check to see if SD Card is mounted
    std::string data = ""; // Create output string
    data = read_file(u"", "NEWER_FILE.TXT"); // read_file()
    if(data != "0") { // Success is anything but 0
        // Logic if data is present
    } else {
        // Logic if there's no data
    }
} else { // Else, check_sd_card() returned false
    // Logic else SD Card is NOT mounted
}

```

Write File

To write to a file the `write_line()` or `write()` function from the [File](#) class can be used. The helper function below will return true if the class object `File` appends to the target file successfully. Success from this append function will have a `is_valid()` function which will return a 0, other values means failure. Inputs for file path and directory name. Again the input for the file path is a `std::filesystem::path` which can be UTF-16 string literal. The root of the SD Card is `u""` and any directory beyond that is `u"DIRECTORY/SUB_DIRECTORY"`

ui_newapp.cpp

```
bool NewAppView::write_file(const std::filesystem::path& path, std::string name, std::string data) {
    File file; // Create File object
    auto success = file.append(path.string() + "/" + name); // Open file
    if(!success.is_valid()) { // 0 is success
        file.write_line(data);
        return true;
    } else {
        return false;
    }
}
```

ui_newapp.cpp

```
if(check_sd_card()) { // Check to see if SD Card is mounted
    std::string data = "Your mother was a hamster!";
    if(write_file(u"", "NEWER_FILE.TXT", data)) { // Success is anything but 0
        // Logic if write was successful
    } else {
        // Logic if write failed
    }
} else { // Else, check_sd_card() returned false
    // Logic else SD Card is NOT mounted
}
```

Rename File or Directory

To rename a file or directory the `rename_file()` function from [File](#) can be used. The helper function below will return true if `rename_file()` succeeds (Success from `rename_file()` returns a 0, other values means failure) and takes two variable inputs for file path and directory name. Again the input for the file path is a `std::filesystem::path` which can be UTF-16 string literal. The root of the SD Card is `u""` and any directory beyond that is `u"DIRECTORY/SUB_DIRECTORY"`

ui_newapp.cpp

```
bool NewAppView::rename_dir_or_file(const std::filesystem::path& path, std::string old_name, std::string new_name) {
    return !(rename_file(path.string() + "/" + old_name, new_name)); // 0 is success
}
```

ui_newapp.cpp

```
// Rename directory or file
if(check_sd_card()) { // Check to see if SD Card is mounted
    if(rename_dir_or_file(u"", "NEW_DIR", "NEWER_DIR")) { // Renames dir or file in root of SD Card, returns 0 if success
        // Logic if rename is successful
    } else { // Else new dir or file renamed failed
        // Logic if rename failed
    }
}
```

```

    }
} else { // Else, check_sd_card() returned false
    // Logic else SD Card is NOT mounted
}

```

Delete File or Directory

To delete a file or directory the `delete_file()` function from [File](#) can be used. The helper function below will return true if `delete_file()` succeeds (Success from `make_new_directory()` returns a 0, other values means failure) and takes two variable inputs for file path and directory name. Again the input for the file path is a `std::filesystem::path` which can be UTF-16 string literal. The root of the SD Card is `u""` and any directory beyond that is `u"DIRECTORY/SUB_DIRECTORY"`

ui_newapp.cpp

```

bool NewAppView::delete_dir_or_file(const std::filesystem::path& path, std::string name) {
    return !(delete_file(path.string() + "/" + name));
}

```

Below is an example on how the `delete_dir_or_file()` function can be used with basic error handling.

ui_newapp.cpp

```

if(check_sd_card()) { // Check to see if SD Card is mounted
    if(delete_dir_or_file(u"", "NEW_DIR")) { // New dir in root of SD Card
        // Logic if file/dir delete was successful
    } else {
        // Logic if file/dir delete was NOT successful
    }
} else { // Else, check_sd_card() returned false
    // Logic else SD Card is NOT mounted
}

```

Wrap Up

Below is example demo of all basic CRUD functions for files and directories.

ui_newapp.hpp

```

#include "ui.hpp"
#include "ui_widget.hpp"
#include "ui_navigation.hpp"
#include "string_format.hpp"

// Add these to include File Class and SD Card helper functions.
#include "file.hpp"
#include "sd_card.hpp"

namespace ui
{
    // Struct that outlines file information
    struct file_entry {
        std::filesystem::path entry_path { };
        uint32_t size { };
        bool is_directory { };
    };

    class NewAppView : public View // App class declaration
    {
    public:

        // Public declarations

```

```

void focus() override; // ui::View function override

NewAppView(NavigationView &nav); // App class init function declaration
std::string title() const override { return "New App"; }; // App title

private:

    // Function declarations

    // Error check
    bool check_sd_card();

    // DIR CRUD
    bool create_dir(const std::filesystem::path& path, std::string name);
    std::vector<file_entry> list_dir(const std::filesystem::path& path);

    // File CRUD
    bool create_file(const std::filesystem::path& path, std::string name);
    bool rename_dir_or_file(const std::filesystem::path& path, std::string old_name, std::string new_name);
    std::string read_file(const std::filesystem::path& path, std::string name);
    bool write_file(const std::filesystem::path& path, std::string name, std::string data);
    bool delete_dir_or_file(const std::filesystem::path& path, std::string name);

    // Widgets
    Console my_console {
        { 1*8, 1*8, 224, 296 }, // Coordinates are: int:x (px), int:y (px), int:width (px), int:height (px)
    };
};

```

ui_newapp.cpp

```

#include "ui_newapp.hpp"
#include "portapack.hpp"
#include <cstring>

using namespace portapack;

namespace ui
{
    void NewAppView::focus() { // Default selection to my_console when app starts
        my_console.focus();
    }

    // Checks SD Card, returns true if Mounted, false if otherwise
    bool NewAppView::check_sd_card() {
        return (sd_card::status() == sd_card::Status::Mounted) ? true : false;
    }

    // Lists all files and directories in path
    std::vector<file_entry> NewAppView::list_dir(const std::filesystem::path& path) {

        // Files and directories list
        std::vector<file_entry> entry_list { };

        // For each entry in the file system's directory
        // Adds files in directories into entry_list{}
        // Directories are inserted in front of files
        for (const auto& entry : std::filesystem::directory_iterator(path, u"*)) {

            // Do not add directories or files starting with '.' (hidden / tmp)
            if (entry.path().string().length() && entry.path().filename().string()[0] != '.') {

                // If file
                if (std::filesystem::is_regular_file(entry.status())) {
                    entry_list.push_back({ entry.path(), (uint32_t)entry.size(), false });
                }

                // Else If directory
                } else if (std::filesystem::is_directory(entry.status())) {
                    entry_list.insert(entry_list.begin(), { entry.path(), 0, true });
                }
            }
        }
    }
}

```

```

        // Other
    } else {
        continue;
    }
}

}
return entry_list;
}

// Creates dir, returns true if successful
bool NewAppView::create_dir(const std::filesystem::path& path, std::string name) { // make_new_directory
    return !(make_new_directory(path.string() + "/" + name)); // and other values
} // will return true

bool NewAppView::delete_dir_or_file(const std::filesystem::path& path, std::string name) { // Deletes
    return !(delete_file(path.string() + "/" + name)); // 0 is success
}

bool NewAppView::create_file(const std::filesystem::path& path, std::string name) {
    File file = { }; // Create File object
    Optional<File::Error> success = file.create(path.string() + "/" + name); // Create File
    return !(success.is_valid()); // 0 is success
}

bool NewAppView::rename_dir_or_file(const std::filesystem::path& path, std::string old_name, std::string new_name) {
    return !(rename_file(path.string() + "/" + old_name, new_name)); // 0 is success
}

std::string NewAppView::read_file(const std::filesystem::path& path, std::string name) { // Read file
    std::string return_string = ""; // String to return
    File file; // Create File object
    auto success = file.open(path.string() + "/" + name); // Open file to read

    if(!success.is_valid()) { // 0 is success
        char one_char[1]; // Read file
        for(size_t pointer = 0; pointer < file.size() ; pointer++) { // Example with file
            file.seek(pointer); // Sets file pointer
            file.read(one_char, 1); // sets character
            return_string += one_char[0];
        }
    } else {
        return "0"; // Basic error
    }
    return return_string;
}

bool NewAppView::write_file(const std::filesystem::path& path, std::string name, std::string data) {
    File file; // Create File object
    auto success = file.append(path.string() + "/" + name); // Open file
    if(!success.is_valid()) { // 0 is success
        file.write_line(data);
        return true;
    } else {
        return false;
    }
}

NewAppView::NewAppView(NavigationView &nav) // Application Main
{
    add_children({
        &my_console, // Add pointers for widgets
    });

    // Enable scrolling
    my_console.enable_scrolling(true);
}

```

```

// Check SD Card
if(check_sd_card()) {
    my_console.writeln("+ SD Card is Mounted");
} else {
    my_console.writeln("- SD Card is NOT Mounted");
}

// New directory
if(check_sd_card()) {
    if(create_dir(u"", "NEW_DIR")) {
        my_console.writeln("+ New directory created");
    } else {
        my_console.writeln("- New directory FAILED");
    }
} else {
    my_console.writeln("- New directory FAILED");
}

// List Directory
std::string dir_contents = "";
std::vector<file_entry> files = { };
if(check_sd_card()) {
    files = list_dir(u "");
    if(files.size()) {
        dir_contents += "+ dir SD Card: ";
        for (const auto& f : files) {
            dir_contents += f.entry_path.string() + ", ";
        }
    } else {
        my_console.writeln("- dir SD Card FAILED");
    }
} else {
    my_console.writeln("- dir SD Card FAILED");
}
//my_console.writeln(dir_contents);
my_console.writeln("+ dir SD Card: " + std::to_string(files.size()) + " items");

// Rename directory
if(check_sd_card()) {
    if(rename_dir_or_file(u"", "NEW_DIR", "NEWER_DIR")) {
        my_console.writeln("+ Directory renamed");
    } else {
        my_console.writeln("- Directory renamed FAILED");
    }
} else {
    my_console.writeln("- Directory renamed FAILED");
}

// Delete file or directory
if(check_sd_card()) {
    if(delete_dir_or_file(u"", "NEWER_DIR")) {
        my_console.writeln("+ New directory deleted");
    } else {
        my_console.writeln("- New directory deleted FAILED");
    }
} else {
    my_console.writeln("- New directory deleted FAILED");
}

// New file
if(check_sd_card()) {
    if(create_file(u"", "NEW_FILE.txt")) {
        my_console.writeln("+ New file created");
    } else {
        my_console.writeln("- New file FAILED");
    }
} else {
    my_console.writeln("- New file FAILED");
}

// Rename file

```

```

if(check_sd_card()) {
    if(rename_dir_or_file(u"", "NEW_FILE.txt", "NEWER_FILE.TXT")) { // Check to see if SD Card is
        my_console.writeln("+ File renamed"); // Renames file, returns true
    } else { // If file renamed succeeded
        my_console.writeln("- File renamed FAILED"); // Else new file renamed failed
    }
} else { // Else, check_sd_card() returned false
    my_console.writeln("- File renamed FAILED");
}

// Write file
if(check_sd_card()) { // Check to see if SD Card is mounted
    std::string data = "Your mother was a hamster!";
    if(write_file(u"", "NEWER_FILE.TXT", data)) { // Success is anything but 0
        my_console.writeln("+ Write File"); // Write data to my_console
    } else {
        my_console.writeln("- Write file FAILED");
    }
} else { // Else, check_sd_card() returned false
    my_console.writeln("- Write file FAILED");
}

// Read file
if(check_sd_card()) { // Check to see if SD Card is mounted
    std::string data = ""; // Create output string
    data = read_file(u"", "NEWER_FILE.TXT"); // read_file()
    if(data != "") { // Success is anything but 0
        my_console.writeln("+ Read file: " + data); // Write data to my_console
    } else {
        my_console.writeln("- Read file FAILED");
    }
} else { // Else, check_sd_card() returned false
    my_console.writeln("- Read file FAILED");
}

// Delete file or directory
if(check_sd_card()) { // Check to see if SD Card is mounted
    if(delete_dir_or_file(u"", "NEWER_FILE.TXT")) { // New dir in root of SD Card
        my_console.writeln("+ New file deleted");
    } else {
        my_console.writeln("- New file deleted FAILED");
    }
} else { // Else, check_sd_card() returned false
    my_console.writeln("- New file deleted FAILED");
}

// Done
my_console.writeln("+ Demo Complete");
}

```

Access Radio Hardware

About

Now we're going to dive into something a little more complex and bring radios into the mix. If you're not familiar with the LPC43xx please read up on the [Firmware Architecture](#) before continuing.

So far we've only been dealing with application code with is ran on the M0 of the LPC43xx. Now we're going to start working with the baseband side of the codebase which is ran on the LPC43xx's M4 processor. Both of these processors use 8k worth of shared memory from 0x1008_8000 to 0x1008_a000 to pass messages to and from each other. The M0 controls ALL operations of the portapack while the M4 mostly handles the DSP and radio functions.

Complexity aside with the two processors, accessing the HackRF's radio hardware has been simplified with helper classes such as the [TransmitterModel](#) and [ReceiverModel](#). Both of these classes interface with the M4 baseband processes and gives us a more piratical way to control the radio.

Other classes and structs such as [baseband_api](#) and [SharedMemory](#), also bridge the gap between the M0 and M4. Even though the M4's primary responsibility is to handle DSP with the radio hardware the M0 can still be used to decode data. For example, classes found in `firmware/application/protocols/` like [encoders](#) still send data too and from the two processors but also encodes and decodes messages at the higher level protocols.

TX

The code bellow is an example OOK TX application using [TransmitterModel](#), [encoders](#), and [baseband](#).

ui_newapp.hpp

```
....

// Include TransmitterModel
#include "transmitter_model.hpp"

namespace ui
{
    class NewAppView : public View // App class declarati
    {
    public:

        ....

    private:

        ....

        void start_tx(std::string& message); // Function declarati
        void stop_tx();
        void on_tx_progress(const uint32_t progress, const bool done);

        MessageHandlerRegistration message_handler_tx_progress { // MessageHandlerRegi
            Message::ID::TXProgress, // Message::ID::TXPro
            [this](const Message* const p) { // code from baseband
                const auto message = *reinterpret_cast<const TXProgressMessage*>(p); // an uint32_t progre
                this->on_tx_progress(message.progress, message.done); // TX progress has be
            }
        };
    };
}
```

ui_newapp.cpp

```
....

// Include encoders and baseband
#include "encoders.hpp"
#include "baseband_api.hpp"

using namespace portapack;

namespace ui
{
    void NewAppView::start_tx(std::string& message) // Message input as "101101"
    {
        size_t bitstream_length = make_bitstream(message); // Function from encoders.hpp. Encodes t
        // sets message to TX data pointer via.
        // uint8_t * bitstream = shared_memory.k
        // on line 34 of encoders.cpp and return

        transmitter_model.set_tuning_frequency(433920000); // Center frequency in hz
    }
}
```

```

transmitter_model.set_sampling_rate(OOK_SAMPLERATE); // (2280000) Value from encoders.hpp
transmitter_model.set_rf_amp(true); // RF amp on
transmitter_model.set_baseband_bandwidth(1750000); // Bandwidth
transmitter_model.enable(); // Radio enable

baseband::set_ook_data( // ASK/OOK TX function
    bitstream_length, // Length of message
    OOK_SAMPLERATE / 1766, // Symble period (560us), Sample Rate /
    4, // Repeat transmissions
    100 // Pause symbles
);
}

void NewAppView::stop_tx() // Stop TX function
{
    transmitter_model.disable(); // Disable transmitter_model

    // Add UI logic to let the user know the TX has stoped
}

NewAppView::NewAppView(NavigationView &nav) // Application Main
{
    baseband::run_image(portapack::spi_flash::image_tag_ook); // M4 processor is being told to run pro
    // found in the firmware/baseband/ folder
    // then reset after this command.

    // UI widget logic and calls to
    // start_tx() goes here.
}

void NewAppView::on_tx_progress(const uint32_t progress, const bool done) // Function logic for when
{ // sends a TXProgressMessage
    if(done) {
        stop_tx();
    } else {
        // UI logic, update ProgressBar with progress var
    }
}
}

```

RX

Building from the example code for TX lets talk about how the baseband processes are started on the M4. The application code on the M0 uses the baseband api `baseband::run_image` to tell the M4 to run a process. The baseband images are defined in [spi_image.hpp](#) as the struct `image_tag_t`. These structs have a 4 char array tag being used as an ID. Below is an example `image_tag_t` for AFSK RX.

```
constexpr image_tag_t image_tag_afsk_rx { 'P', 'A', 'F', 'R' };
```

Under [firmware/baseband/CMakeLists.txt](#) the following code snippet shows how the baseband processes are linked to the images defined in [spi_image.hpp](#).

```

### AFSK RX
set(MODE_CPPSRC
    proc_afskrx.cpp
)
DeclareTargets(PAFR afskrx)

```

In `firmware/baseband`, process or "proc" code for the M4 processor like [proc_afskrx.cpp](#) for example can be found here. These proc classes are ran by [BasebandThread](#). All proc classes inherit [BasebandProcessor](#) and must include the parent functions.

baseband_processor.hpp

```

#ifndef __BASEBAND_PROCESSOR_H__
#define __BASEBAND_PROCESSOR_H__

#include "dsp_types.hpp"
#include "channel_stats_collector.hpp"
#include "message.hpp"

class BasebandProcessor {
public:
    virtual ~BasebandProcessor() = default;           // Constructor

    virtual void execute(const buffer_c8_t& buffer) = 0; // DSP code for TX/RX, shared_memory messages ca
                                                         // M0 application code from this function.

    virtual void on_message(const Message* const) { }; // Shared_memory messages from M0 application co

protected:
    void feed_channel_stats(const buffer_c16_t& channel);

private:
    ChannelStatsCollector channel_stats { };
};

```

Now that we have a better idea how M0 can drive the M4 lets talk about the Messaging between the two processors. The [Message](#) class found under `firmware/common/`. Common code is used both by application (M0) and baseband (M4). Messages are handled by EventDispatcher found in [event_m4.cpp](#) for the baseband code and [event_m0.cpp](#) for the application code. Within the same file [firmware/commen/message.hpp](#) you can find definitions for spacific message classes and ID. Bellow is an example message class for AFSK RX.

message.hpp

```

class Message {
public:
    static constexpr size_t MAX_SIZE = 512;

    enum class ID : uint32_t {
        /* Assign consecutive IDs. IDs are used to index array. */

        ....

        AFSKRxConfigure = 22,
        AFSKData = 47,

        ....
    };
}

....

// Application Messages (M0) -> Baseband (M4)
class AFSKRxConfigureMessage : public Message {
public:
    constexpr AFSKRxConfigureMessage(
        const uint32_t baudrate,
        const uint32_t word_length,
        const uint32_t trigger_value,
        const bool trigger_word
    ) : Message { ID::AFSKRxConfigure },
        baudrate(baudrate),
        word_length(word_length),
        trigger_value(trigger_value),
        trigger_word(trigger_word)
    {
    }
}

```

```

    const uint32_t baudrate;
    const uint32_t word_length;
    const uint32_t trigger_value;
    const bool trigger_word;
};

// Baseband Messages (M4) -> Application (M0)
class AFSKDataMessage : public Message {
public:
    constexpr AFSKDataMessage(
        const bool is_data,
        const uint32_t value
    ) : Message { ID::AFSKData },
        is_data { is_data },
        value { value }
    {
    }

    bool is_data;
    uint32_t value;
};

```

[SharedMemory](#) found in `firmware/common/` is used to pass data inbetween the application code (M0) to the baseband code (M4). Below is an example from [proc_afskrx.cpp](#) on how data is sent back to the application [AFSKRxView](#).

proc_afskrx.cpp

```

#include "portapack_shared_memory.hpp"

void AFSKRxProcessor::execute(const buffer_c8_t& buffer) {

    ....                                // RX Logic

    shared_memory.application_queue.push(data_message); // data_message is an AFSKDataMessage object

    ....                                // MORE RX Logic

};

```

Continuing to use the same AFSK RX proc code above, below is an example of an RX AFSK application.

ui_newapp.hpp

```

....

// Include ReceiverModel
#include "receiver_model.hpp"

namespace ui
{
    class NewAppView : public View // App class declaration
    {
    public:

        ....

    private:

        ....

        void start_rx(); // Function declarations
        void stop_rx();
        void on_data();
        MessageHandlerRegistration message_handler_packet { // MessageHandlerRegistratio
            Message::ID::AFSKData, // relays messages to your a
            [this](Message* const p) { // Every time you get a AFSK

```

```

        const auto message = static_cast<const AFSKDataMessage*>(p); // on_data() function will be
        this->on_data(message->value, message->is_data);
    }
};

```

```

};

```

ui_newapp.cpp

```

....

#include "modems.hpp"
#include "audio.hpp"
#include "string_format.hpp"
#include "baseband_api.hpp"
#include "portapack_persistent_memory.hpp"

using namespace portapack;
using namespace modems;

namespace ui
{
    void NewAppView::start_rx() // Start RX function
    {
        auto def_bell202 = &modem_defs[0]; // Bell202 baud rate
        persistent_memory::set_modem_baudrate(def_bell202->baudrate); // Set RX modem to 1200 baud

        serial_format_t serial_format; // Declare packet format for RX
        serial_format.data_bits = 7; // Bit length
        serial_format.parity = EVEN; // Even or odd parity bit
        serial_format.stop_bits = 1; // Stop bit
        serial_format.bit_order = LSB_FIRST; // LSB or MSB first
        persistent_memory::set_serial_format(serial_format); // Set RX packet format

        baseband::set_afsk(persistent_memory::modem_baudrate(), 8, 0, false); // Baud rate, word length,
        // parity, and stop bits

        receiver_model.set_tuning_frequency(433920000); // Center frequency in hz
        receiver_model.set_sampling_rate(3072000); // Sampling rate
        receiver_model.set_baseband_bandwidth(1750000); // Bandwidth
        receiver_model.set_modulation(ReceiverModel::Mode::NarrowbandFMAudio); // Modulation
        receiver_model.enable(); // Start RX

        audio::set_rate(audio::Rate::Hz_24000); // Play RX audio to headphones
        audio::output::start();
    }

    void NewAppView::stop_rx() // Stop RX function
    {
        audio::output::stop(); // Stop Audio
        receiver_model.disable(); // Stop RX
        baseband::shutdown(); // Stop M4 proc process
    }

    NewAppView::NewAppView(NavigationView &nav) // Application Main
    {
        baseband::run_image(portapack::spi_flash::image_tag_afsk_rx); // M4 processor is being told to run
        // found in the firmware/baseband/ folder
        // then reset after this command.

        // UI widget logic and calls to start_rx()
        // and stop_rx() goes here.
    }

    void NewAppView::on_data(uint32_t value, bool is_data) // Function logic for when the message
    { // sends a AFSKData.
        if(is_data) {
            // RX data handling Logic
        }
    }
}

```

```
}  
}
```

Interpret Guru meditation crashes

Sometimes your program is having a bad day, and it's simply crashing the whole machine. We added an error screen for developers who don't have a [Black Magic Probe](#) sitting around and ready to use. You should still be able to narrow down the location where the error is coming from.



Information On Screen

- The first line already contains the first very important piece.
 - M0: the crash occurred in the application firmware
 - M4: the crash occurred in the baseband firmware
- The Hint can be
 - Hard Fault: indicates an invalid operation. (eg. invalid memory access)
 - MemManage: indicates a memory fault
 - BusFault: indicates a bus fault
 - UsageFault: indicates a usage fault
 - some other text like 'NoImg' or 'BBRunning' (see usages of 'chDbgPanic(const char *)')
- Registers (only visible in case of a Hard Fault)
 - r0-r3 & r12: The values of the registers at the time of the fault.
 - lr: The link register contains the return address of the calling method. Use this value only if the process counter is unusable.
 - pc: The process counter is the location of the current instruction. Use this value in the next step.

Determining the location in C/C++ source

- Run the following command to get the assembly at the location.
 - Replace 0xd440 with your pc/lr value
 - use firmware/application/application.elf only for M0 errors. Use the currently loaded baseband image for M4 errors. (eg. firmware/baseband/baseband_adsbrx.elf)

If you use Arch and see this error: error while loading shared libraries: libncurses.so.5, install [ncurses5-compat-libs](#) from AUR.

```
dev@ubuntu:~$ cd build
dev@ubuntu:~$ arm-none-eabi-gdb --q -ex="x/3i 0xd440" --batch firmware/applicat
0xd440 <luaD_protectedparser>:      push      {r4, r5, r6, lr}
0xd442 <luaD_protectedparsencurses5-compat-libsr+2>:      movs      r5, #0
0xd444 <luaD_protectedparser+4>:      sub       sp, #32
```

- Or run the following command to disassemble the whole image.

```
dev@ubuntu:~$ cd build
dev@ubuntu:~$ arm-none-eabi-objdump --source firmware/application/application.e
```

- then inspect the file firmware/application/application.objdump to get the full picture.

```
[...]
int luaD_protectedparser (lua_State *L, ZIO *z, const char *name) {
    d440:      b570      push      {r4, r5, r6, lr}
    struct SParser p;
    int status;
    p.z = z; p.name = name;
    luaZ_initbuffer(L, &p.buff);
    d442:      2500      movs      r5, #0
int luaD_protectedparser (lua_State *L, ZIO *z, const char *name) {
    d444:      b088      sub       sp, #32
    status = luaD_pcall(L, f_parser, &p, savestack(L, L->top), L->errfunc);
    d446:      6883      ldr       r3, [r0, #8]
[...]
```

M0 Stack Dump

After a crash, pressing the DFU button will display the M0 core's stack contents, which can be paged up & down. All stack words are shown beginning with the first stack word that has been used since powering up the PortaPack (which may not correspond with the current stack pointer). If the LR value is known, stack words containing the value matching the LR where the fault occurred will be highlighted in yellow. Stack words containing values that might possibly be addresses in the code region are highlighted in white.

More useful Information

- [How to debug a HardFault on an ARM Cortex-M on interrupt.memfault.com](#)
- You can read the related PR with instructions [here](#)

Theme-system

There is a Theme system in PP, so each app can use the same color scheme, that can be changed by the user.

Using theme system

It is important to use this system to give users consistent UI. Have to mention there are use cases when you must use specific colors, like in a waterfall, then do it.

To use the theme system, just include **theme.hpp**, and instead of using `Color::grey()` or similar colors, use the corresponding theme object. For example the labels usually used LightGrey color. Now you can use the template's light foreground color. `Theme::getInstance()->fg_light`

The theme system uses (mostly) styles, that starts with `fg_` or `bg_`. These can be light, medium, dark, darker, darkest. FG stands for foreground, BG stands for background. This is to specify if the foreground is important to you, or the background. So if it is important, that the foreground of a button be light, then use the `fg_light`. then the foreground will be light colored (like light grey) but the background for it will be something that has good readability.

There are specific colors to indicate "error", "warning", or "ok". These are: `error_dark`; `warning_dark`; `ok_dark`; There are also specific color styles, like `fg_red`, `fg_green`, `fg_yellow`, `fg_orange`, Use these, because theme can override the color of the red, so within a "red theme" (where the backgrounds are red too) the 255,0,0 color could be different for better contrast. But if you use `Color::red()` it may not be as readable as the theme's `fg_red`.

Also if you use theme specific colors, like `fg_red->foreground`, you should use it's background counterpart. (in the example it would be `fg_red->background`).

The app's background is usually `bg_darkest->background`.

You don't need to use the styles as a style object! For labels you can specify the foreground color only.

`Theme::getInstance()->fg_light->foreground` or for the app or widget's background you can use

`Theme::getInstance()->bg_darkest->background`

Creating a new theme

Step 1: open `theme.hpp` and add a new class derived from `ThemeTemplate`:

```
class ThemeMyNewTheme : public ThemeTemplate {
public:
    ThemeMyNewTheme ();
};
```

Step 2: add it to the enum (`ThemeId`) within the `Theme` class, BEFORE `MAX`.

```
enum ThemeId {
    DefaultGrey = 0,
    Yellow = 1,
    Aqua = 2,
    Green = 3,
    Red = 4,
    MyNewTheme = 5,
    MAX
};
```

Step 3: In the `theme.cpp` create a case block for your new enum and class in the `Theme::SetTheme()` function.

```
void Theme::SetTheme(ThemeId theme) {
    if (current != nullptr) delete current;
    switch (theme) {
        case MyNewTheme :
            current = new ThemeMyNewTheme ();
```



```

        break;
    ....
    case DefaultGrey:
    default:
        current = new ThemeDefault();
        break;
    }
}

```

Step 4: Implement your class's constructor, and create each element! If you forget one, then your theme will crash the PP! Best practice is to copy an exists one and modify the color defined in it.

Step 5: Always check your theme everywhere. Open each app, and look for missing or unreadable texts.

Debug with serial

You can only print things to debug for this. If you wanna set break point, checking registers etc, use another debug method.

ref. and code: <https://github.com/portapack-mayhem/mayhem-firmware/pull/2111/files>

usage

1. Include the async msg header in your file

```
#include "usb_serial_asyncmsg.hpp"
```

if err happend after including it, maybe try not to include it but still using it (we are trying to settle the headers to prevent circuit including but it need sometime)

- 2 print things in your code and it will appear on serial.

```

void FileManBaseView::push_dir(const fs::path& path) {
    std::string test_string = "test string:";
    UsbSerialAsyncmsg::asyncmsg(test_string);
    std::string test_string111 = "abcdefghi";
    UsbSerialAsyncmsg::asyncmsg(test_string111);

    std::string test_path_label = "test_path:";
    UsbSerialAsyncmsg::asyncmsg(test_path_label);
    UsbSerialAsyncmsg::asyncmsg(path);

    std::string test_vec_label = "test_vector:";
    UsbSerialAsyncmsg::asyncmsg(test_vec_label);

    std::vector<uint32_t> test_vector;
    test_vector.push_back(1);
    test_vector.push_back(2);
    test_vector.push_back(3);
    UsbSerialAsyncmsg::asyncmsg(test_vector);

    std::string test_num_label = "test_num:";
    UsbSerialAsyncmsg::asyncmsg(test_num_label);

    uint8_t test = 254;
    UsbSerialAsyncmsg::asyncmsg(test);

    if (path == parent_dir_path) {
        pop_dir();
    } else {
        current_path /= path;
        saved_index_stack.push_back(menu_view.highlighted_index());
        menu_view.set_highlighted(0);
        reload_current(true);
    }
}

```

why not printing anything?

- make sure async msg lock is disabled.

```
asyncmsg enable
```

- make sure serial connected

```
help
```


- MayhemHub currently not support real-time serial command communication. Use others.

linking error

It's possible that the object you are printing isn't implemented yet (e.g. color obj). If you would like, feel free to implement it by editing `usb_serial_asyncsg.cpp` and `hpp` files.

I2C drivers

I2C external device driver integration

 Important

Always check the wiring, because you can kill your devices!

How the I2CManager works

On boot, the system will do a full I2C bus scan, and try to find each device it's driver. If a driver is not found, it won't look for it again. The scan is repeated only on main screen and by any app that requests it. If your app requires the periodical scan for new devices you can enable it but disable it in the destructor! These scans will detect new devices and device removals. Devices will be removed when the I2C communication to it's address fails multiple times in a row. If the device is removed, but found again (like plugged in again) the driver will be started again for it. Any app can get a pointer for any driver, and call it's public functions, so it is not forced to just use the `update()` polling method. But for this, you should write it's own app. The app is preferred to be external app.

Basics

The namespace is `i2cdev`, there is everything you'll need. Currently all drivers are under the `firmware/common` folder. You can take the `i2cdev_bmx280` as an example. It has tons of comments that will help you. Important files:

- `i2cdevmanager.hpp` - the core of everything. It describes the `I2cDev`, and the `I2CDevManager`.
- `i2cdevlist.hpp` - here you must put your new driver.
- `i2cdev_XXXXXX.hpp` + `cpp` - the driver itself.

Steps needed

1. Get a good name for your module. From now we will use `I2CDEVMDL_EXAMPLEDEV`
2. Insert a new enum element into the `i2cdevlist.hpp` at the END of the enum `I2C_DEVMDL`. Also create define for the dev's I2C address (for all if it has multiple). This should be `I2CDEV_EXAMPLEDEV_ADDR_1`, `I2CDEV_EXAMPLEDEV_ADDR_2`, ...
3. Create the new `i2cdev_exampledev.hpp` and `i2cdev_exampledev.cpp` files. Use the namespace `i2cdev`!

4. Create the class for the device named `I2cDev_ExampleDev`, and derive it from `I2cDev`.
5. Override the `bool init(uint8_t addr)` function. This function must check the address it got if the module uses it. If yes, then try to init the device. Query for any special registers that you can identify the dev, to make sure it is THAT device you are writing the driver for (since different devices share the same address sometimes). If THAT is your device, then set it up and get it ready to go. Also you must return true if the driver is ok. If there is ANY error, you must return false. In this function you MUST set the variable `addr` to the given address (`addr = addr_;`). You MUST set the model variable to the new enum you just created. Also you need to set the `query_interval` value, how often does your device needs to be polled for new data. Don't put here too small value, since it'll slow down the whole system. The value is in seconds. You can use the `I2cDev::i2c_read()` and `i2c_write()` functions or one of the helper functions to read / write from the bus. `init()` must be as fast as possible.
6. Override the `void update()` function. Here you query your device for new data. If you got any value, then you broadcast it system wide. (see next step). `update()` must be as fast as possible. Try not to delay there. So please don't run calibration code that took 1000 samples and 10 seconds.
7. Select the message you want to use. The current system wide messages are in the `message.hpp`. If you don't find any that fits your need, you can create a new one. (try to avoid that).
8. In your `update()` code just create a new message variable you have selected, take for example the `EnvironmentDataMessage msg{temp, hum, pressure};` fill it with your data, and send it to the system with the `EventDispatcher::send_message(msg);` function.
9. Include your `hpp` file (`i2cdev_exampledev.hpp`) in the `i2cdevmanager.cpp`. Then add it to the `found()` function. There check if the currently found device's address matches any you can handle, and if yes, create an instance of your class (`item.dev = std::make_unique<I2cDev_ExampleDev>()`), and try to call `init()`. If it fails, set the `item.dev = nullptr` so other drivers can try to work with it.
10. Update apps / create apps to handle your data.
11. Test, test and test.
12. Clean up your code. Remove any unneeded things, use the less needed variable sizes. Try to use the less FW space and RAM. There is not so much ram assigned to this task.

Debug

You can use the `Debug / ExtSensor` app to see what I2C devices are found by the system. You can use the `usb_serial_asyncmsg.hpp` to send debug messages to USB serial. (before you must send "asyncmsg enable").

App development for devices

`I2CDevManager::manual_scan()` will start an one time scan for new (or removed) devices.
`I2CDevManager::set_autoscan_interval()` will start a periodic search for new devices. DON'T FORGET TO SET IT TO 0 when you finished or your app is closed! `I2CDevManager::get_dev_by_addr()`, `I2CDevManager::get_dev_by_model()` will give you a pointer to the device. Preferred to use the `get_dev_by_model()`, since with that you'll know what that pointer needs to be casted (it'll return a generic `I2cDev`, that you derived your driver from). `I2CDevManager::get_dev_list_by_model()` and `I2CDevManager::get_gev_list_by_addr()` returns a vector of the models it discovered (that HAS WORKING DRIVER) and all the addresses it sees on the bus (even those without a driver). When the scan detects any change in the device list it'll send a "I2CDevListChangedMessage" system wide message.

make_airlines_db.py

This tool creates [airlines.db](#). This database is used to find the *airline name* and *country* based on the three-letter ICAO code. This database is currently only used by ADS-B receiver application.

Source for this database is: <https://raw.githubusercontent.com/kx1t/plane-fence-airlinecodes/main/airlinecodes.txt>

Build database

- Download the latest version from location above.

- Download script [make_airlines_db.py](#).
- Put them in the same folder and run script. Note: Python 3 required.
- Copy database to /ADSB folder on sdcard.

make_icao24_db.py

This tool creates [icao24.db](#).

This database is used to find aircraft-related information based on the 24-bit ICAO transponder code. This database is currently only used by ADS-B receiver application.

Source for this database is: <https://opensky-network.org/datasets/metadata/aircraftDatabase.csv>

Build database

- Copy file from: <https://opensky-network.org/datasets/metadata/aircraftDatabase.csv>
- Run Python 3 [script](#): `./make_icao24_db.py`
- Copy file to /ADSB folder on SDCARD

world_map.bin

The Release package includes a map, however you can create your own following simple rules:

- `world_map.jpg` should be square (i.e. 32768 x 32768 px)
- The projection of the map is Mercator based on the World Geodetic System (WGS) 1984 geographic coordinate system (datum)
- Longitude and latitude cover the map from -180 to 180 and -85 to 85, respectively. Note that latitude does not go up to 90 (since most available maps do not include the 5 initial degrees)
- Center of the map should be 0-degrees latitude and 0-degrees longitude (where the prime meridian and equator intersect)

With those conditions fulfilled, place the input map in: `sdcard/ADSB/world_map.jpg` and then run `firmware/tools/generate_world_map.bin.py`. Your map will be generated on: `sdcard/ADSB/world_map.bin`

converter

Splash screen

For converting back and forth between the image and source you can use `xxd`. In a Windows environment you can use this command in the Linux Subsystem installing one of the versions available (i.e. [Ubuntu](#)).

From .bmp to .hpp

```
xxd -i input_image.bmp output.hpp
```

From .hpp to .bmp

```
xxd -r -p input.hpp output_image.bmp
```

Python converter for Splash Screen

There is also a python helper in [firmware/tools/bmp_tools](#)

Firmware icons

The special file `firmware/application/bitmap.hpp` contains all firmware icons. This file should not be edited directly! Instead, new icons should be copied in [firmware/graphics](#) and then run `python [firmware/tools/bitmap_tools/pp_png2hpp.py]` (https://github.com/portapack-mayhem/mayhem-firmware/blob/next/firmware/tools/bitmap_tools/pp_png2hpp.py) <DIRECTORY> `bitmap.hpp` to create `bitmap.hpp`.

For operating that Python3 script, Pillow library is required: `pip install pillow`

make_bitmap.py - Convert a folder contains one or more icon.png to one bitmap.hpp

The `make_bitmap.py` is the traditional helper, well tested. The folder with the icons is given as argument and generates the `bitmap.hpp`. This file needs to be copied to `mayhem-firmware/firmware/application/`. Starting from the `tools/` folder, the command is:

```
python3 bitmap_tools/make_bitmap.py ../graphics/ && mv bitmap.hpp ../application/bitmap.hpp
```

The icon size needs to be a multiple of 8. The generated icon is black/white.

bitmap_arr_reverse_decode.py - Convert bitmap array to icon.png

The `bitmap_arr_reverse_decode.py` takes an array from the `bitmap.hpp` and convert it back to a png.

pp_png2hpp.py - Convert both ways

The `pp_png2hpp.py` is based on the previous scripts, as all in one solution.

With the `--hpp bitmap.hpp` file and the `--graphics /folder_to/png_icons/` arguments, this script will generate a `bitmap.hpp`.

Add the `--reverse` argument to generate png icons from a given `bitmap.hpp`.

The reverse function got a parser, for automatic get the filename and size of the image.

Example

From the `tools/` folder:

```
$ ./bitmap_tools/pp_png2hpp.py ../application/bitmap.hpp ../graphics
Converting from png to hpp
From path ../graphics/ to file ../application/bitmap.hpp
Find your bitmap.hpp at ../application/bitmap.hpp
```

To reverse the icons from the `.hpp` to png icons use the parameter `--reverse`. Default is only the mayhem logo, named `titlebar_image`:

```
$ ./bitmap_tools/pp_png2hpp.py ../application/bitmap.hpp /home/lupus/work/hackrf/tmp/ --reverse
Reverse: Converting from hpp to png
Converting icon titlebar_image
```

Add the parameter `--icon <icon name>` to convert another icon or `all` to convert all available icons at once:

```
$ ./bitmap_tools/pp_png2hpp.py ../application/bitmap.hpp /home/lupus/work/hackrf/tmp/ --reverse --icon all
```

Note: It is not a recovery, the icons will be new generated. The new icons are black/white, even if the original icons were transparent.

Online editor

The current icons were designed on [Piskelapp](#). To create or update icons, the current sources ([16x16](#) or [8x8](#)) can be used.

Bitmap reverse decode

When you wanna check your bitmap with human eyes, you can use this script to convert it back to image that human eyes can read.

The script is at https://github.com/portapack-mayhem/mayhem-firmware/blob/next/firmware/tools/bitmap_tools/bitmap_arr_reverse_decode.py which is independent of project compile process.

Dissecting the Temperature Logger

Brief History about the Temperature Logger

Some subsystems inside the Portapack would benefit from some sort of temperature compensation being added into the code, as discussed [in the original Portapack Github repository](#) with its developer.

Such need gave birth to the Portapack temperature logger, accessible from **DEBUG->Temperature** menu options.

Temperature data is provided by the **MAX2837 On-Chip Digital Temperature Sensor**. MAX2837 Datasheet can be [found here](#).

But, if you read the discussion about it on the earlier Github link, the temperature precision on the MAX2837 is too coarse (about +/- 5°C) to be really useful.

This is why jboone, the Portapack creator, finally decided this value had no real use and thus the temperature widget further development / refinement and usage enhancement was dropped.

MAX2837 Temperature Sensor

Monitoring MAX2837 temperature is important because its **VCO** (Voltage Controlled Oscillator) is able to maintain lock only while inside +/- 40°C variation from ambient temperature.

NOTE: The MAX 2837 operating temperature range goes from -40°C to +85°C.

This sensor can be activated and read through the SPI interface. By accessing the **TEMP_SENSE** register we can read the ADC output with a precision of **5 bits**.

The datasheet provides the following reference values:

- TA = +25°C -> 01111 (DEC 15)
- TA = +85°C -> 11101 (DEC 29)
- TA = -40°C -> 00001 (DEC 1)

In short, the temp sensor inside MAX2837 returns a decimal number between 1 and 29 for a -40°C to +85°C temperature range, which would put the **precision on about 4.33°C per each value**.

NOTE: As Jboone stated on the Github link provided above, this is definitely a **coarse precision** value.

Temperature Reading

This is accomplished on `/firmware/application/hw/max2837.cpp` with the following code:

```
reg_t MAX2837::temp_sense() {
    if( !_map.r.rx_top.ts_en ) {
        _map.r.rx_top.ts_en = 1;
        flush_one(Register::RX_TOP);

        chThdSleepMilliseconds(1);
    }

    _map.r.rx_top.ts_adc_trigger = 1;
    flush_one(Register::RX_TOP);

    halPolledDelay(ticks_for_temperature_sense_adc_conversion);

    const auto value = read(Register::TEMP_SENSE);

    _map.r.rx_top.ts_adc_trigger = 0;
    flush_one(Register::RX_TOP);

    return value;
}
```

The above code follows the max2837 datasheet's **Temperature Sensor Readout Through DOUT Pin** procedure, which includes the following steps:

- Enable on-chip temperature sensor by setting address 9 (**RX_TOP**), register 1 (**ts_en**) with value 1.
- Wait a while (suggested 100us to 1ms, Portapack code waits 1ms) for the sensor to stabilize and settle within 5 to 1 °C precision.
- Trigger the ADC conversion by setting **RX_TOP** register 0 (**ts_adc_trigger**) with value 1
- According to the max2837 datasheet, the ADC will acquire the temperature value in 2us time (but as we follow the definition of **ticks_for_temperature_sense_adc_conversion** we find a comment in Portapack's code stating a wait of 25 us -shrug-).
- Get the temperature by reading address 7 (**TEMP_SENSE**)
- Finally, turn off the ADC converter by setting address **RX_TOP**, register **ts_adc_trigger** with value 0

Temperature Logging

The logging code functions can be found at `/firmware/application/temperature_logger.cpp`.

There is an array **uint8_t samples** dimensioned for 128 temperature values. The `sample_interval` is set at 5 seconds between each `read_sample()`.

The m0 events dispatcher **handle_rtc_tick** located at `/firmware/application/event_m0.cpp` taps periodically into the function `second_tick()`. When the number of seconds defined by **sample_interval** is reached, the temperature is measured and pushed into the **samples** array.

Temperature Graphic

The temperature graph widget is defined on `/firmware/application/apps/ui_debug.cpp`.

It should graph the 128 temperature values, stored on the `samples` array. On the bottom right side of the graph, it shows the most recent logged value.

First Analysis

Warning: *Here is when things get weird (for me at least).* The Temperature Widget **seems** to be wrong ("Spoiler": Later I learn it is ok):

- First, the temperature graph had the y axis going from 0°C to 60°C (which at first glance is not aligned with MAX2837 temp data output, going from -40°C to 85°C).

- Secondly, when plotting each temperature value, the original sensor value is processed through the following function: `return -45 + sensor_value * 5;` which seems a bit off, particularly when tested against the Ambient Reference sensor value of 15 (for 25°C) according to the MAX2837 Datasheet.

Testing the Temperature graph

I bypassed the °C conversion functionality in order to watch the **raw decimal value** being returned by the MAX2837 On-Chip Digital Temperature Sensor. My test room temperature is about 22°C

First test

Portapack is on, in iddle mode and the sensor **returns a value of 9** . If converted into °C, such value should correspond to a temperature of about 0 / 1 °C, which **seems to be wrong**. Portapack aluminium case is at about 26.7°C.

Second test

I placed my Portapack in transmit mode, loop-playing a random earlier captured radio sample for about 30 minutes. The exterior aluminium case is at about 31.4°C.

Right after stopping the transmission, the sensor value is 10. We give it about 10 seconds and the value drops to 9. But Portapacks aluminium case is still about 31.4°C.

Third test

I placed my Portapack outside, where it cooled down for about 30 minutes (Temperature outside is about 15°C).

Back inside, I powered it up and got a first sensor value of 7, which would correspond to -10°C. Portapack's case temp was about 15°C

About a minute later the sensor value increased by one, returning 8, which should correspond to -5°C. Portapack's case temp was about 20°C

Second analysis

My observation indicates that MAX2832 temperature sensor is NOT giving out an absolute temp value (It reflects neither the ambient temperature inside the Portapack, nor an on-chip temperature).

At first glance, this sensor value **might** be the relative temperature change (increase) between ambient temperature (Which goes up when Portapack is transmitting, as evidenced on the aluminium case external measurement) and the MAX2837 temperature itself.

So what could be wrong ? Consider the following:

- MAX2837 Datasheet may be wrong
- I am not correctly interpreting the MAX2837 description for the On-Chip Digital Temperature Sensor
- Portapack's code might have a bug and reading are wrong

After more code delving, a conclusion is reached: MAX2837 documentation is failing to correctly explain the On-Chip Digital Temperature Sensor values.

MAX2837 Temperature Sensor "redefined"

Apparently, the sensor returns the temperature difference between ambient temperature and MAX2837 temperature!

Differences between H1 and H2

The **H1** is the [original](#) design, which is well documented and in the public domain.

PortaPack **H2** is a modification based on the last (from an unknown origin, no public design documentation available) incorporating some enhancements:

- Bigger screen
- Different navigation buttons
- Internal battery
- Battery charging circuitry
- Better internal Speaker integration

Note: *There are several H1 hybrid models on the global market including **some** of the H2 enhancements but keeping the H1 selector wheel. Some of these H1 hybrids have names similar to **2020 H1 Model, H1 plus, H2+***

Audio

Both feature a **headphone connector** and the **AK4951 AUDIO / CODEC IC** ([datasheet](#)).

The AK4951 includes separate pins for speaker and headphone output:

- Pins 20 (**SPP**) and 19 (**SPN**) labeled as **SPeaker Positive** and **SPeaker Negative** (1W 8 ohm)
- Pins 22 (**HPL**) and 23 (**HPR**) as in (**HeadPhones Left, HeadPones Right**) (16 ohm)

Upon close inspection, several differences arise:

H1

Only the soldering pads for an internal speaker connector header are present. The speaker +/-GND signals on the pad are routed directly into AK4951 pins.

NOTE: *There is no extra amplifier IC between the speaker pad and the AK4951. The AK4951 includes a speaker amplifier capable of 1W, when powered @5v.*

You can solder a laptop / tablet speaker in H1 speaker pads. In order to enable it, you will need to install MAYHEM firmware, where you can also configure under **OPTIONS -> INTERFACE** the addition of a **SPEAKER** button on your top status bar.

H2

PortaPack H2 circuitboard includes a connector header for easily adding an internal speaker.

The **AK4951 IC** has been rotated by 180° and placed more into the middle of the circuit board, allowing for the addition of an extra IC: The ChipStar brand **CS8122S** ultra-low EMI, filter-free, class-D amplifier, for driving the speaker output with up to 3W.

The speaker audio header is switched on/off by the headphone female connector, which includes the usual mechanical switch, detecting the insertion of a male plug in it.

This is a non trivial difference, compared against H1 design where the speaker out pads are routed to the actual speaker out pins on the AK4951. On H2, the audio comes from the Headphones output pins of that IC.

Since the firmware is shared between H1 and H2, on PortaPack H2 you end up powering **TWO** speaker amplifiers: The one inside AK4951 CODEC IC, and the extra 3W speaker amplifier.

H2+ [Variations](#)

At least some of the PortaPack "H2+" variations have the following audio differences:

A **WM8731** audio CODEC may be installed in place of the **AK4951**. The WM8731 IC has no speaker amplifier, for one difference.

An **INS8002E** or **LTK8002D** audio amplifier may be installed in place of the **CS8122S**.

On some H2+ variations, audio volume is low, and inserting a headphone plug does not electrically disable the speaker output (without [additional hardware modifications](#)).

Power Supply

H1

There is no internal battery on your H1 PortaPack: You will need to power it up externally, from your computer or an USB powerbank.

H1 PortaPack design does NOT include any battery charging circuit, nor a provision for a power on/off switch.

Some enthusiasts managed to MOD their H1's by adding a standard usb powerbank management circuit board with the corresponding Li-ion battery and placing a manual power switch on their cases.

H2 and "H2+"

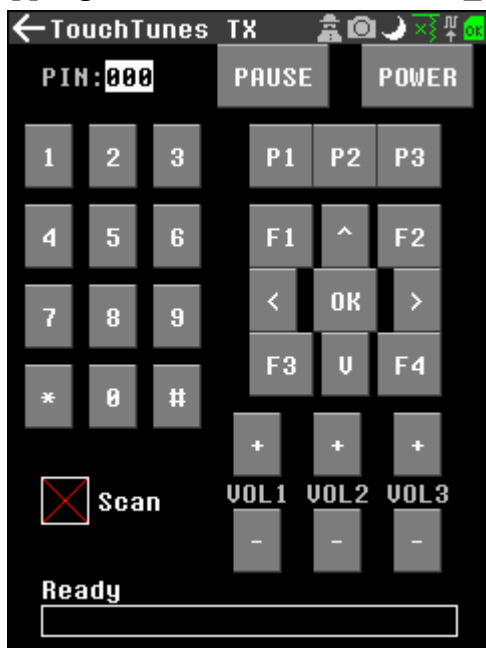
H2 includes an internal Li-Ion battery, standard battery charging / management IC (located at the side of the speakerphone connector) and power switch.

The power switch / battery level feature were provided by IP5306 or similar module.

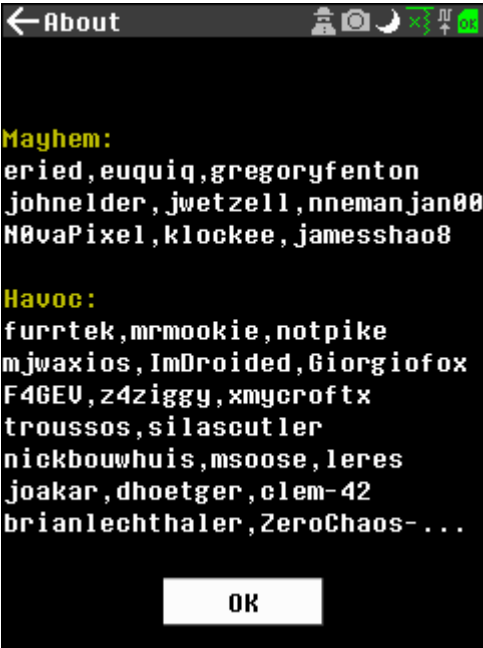
UI Screenshots

Resources for use on the documentation:

[[img/screenshots/Transmit_TouchTune.png]]



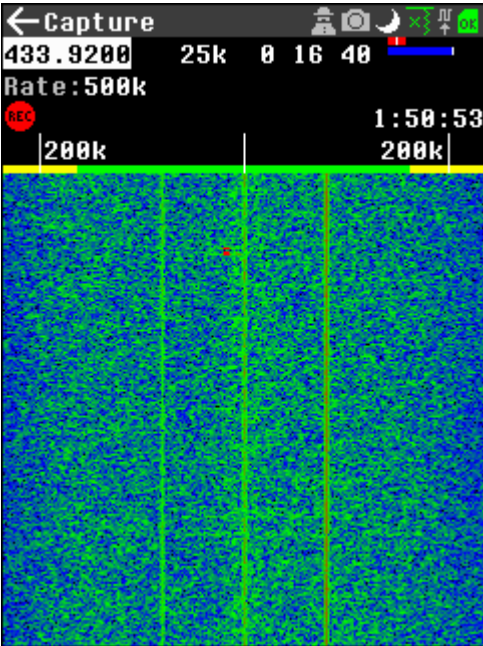
[[img/screenshots/About.png]]



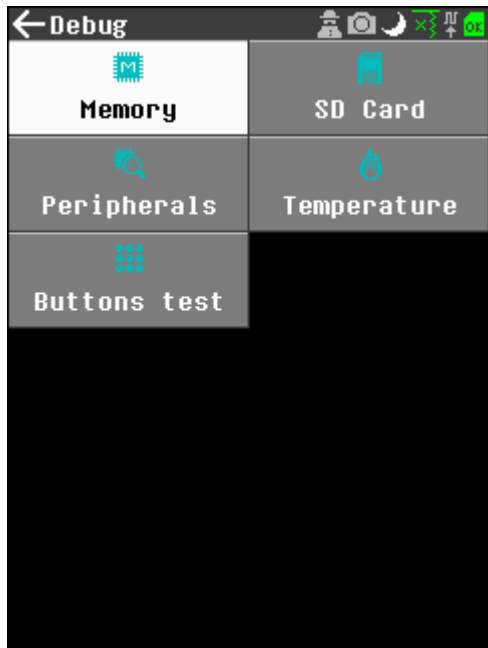
[[img/screenshots/Calls.png]]



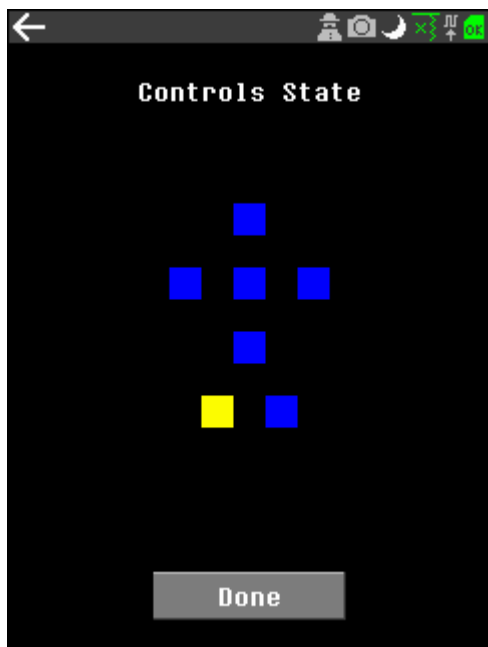
[[img/screenshots/Capture.png]]



[[img/screenshots/Debug.png]]



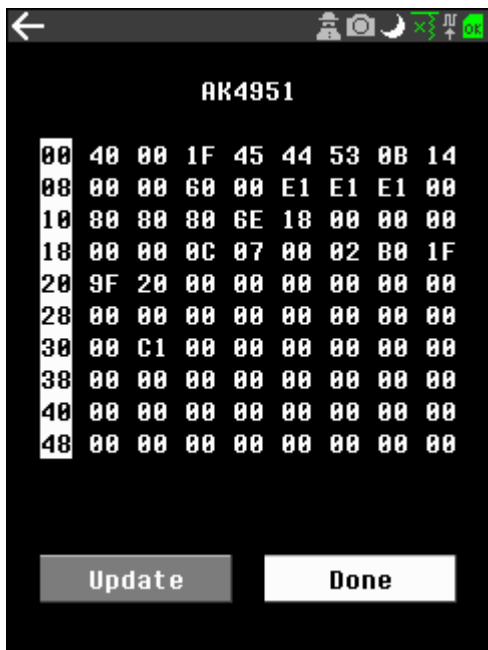
[[img/screenshots/Debug_Buttons test.png]]



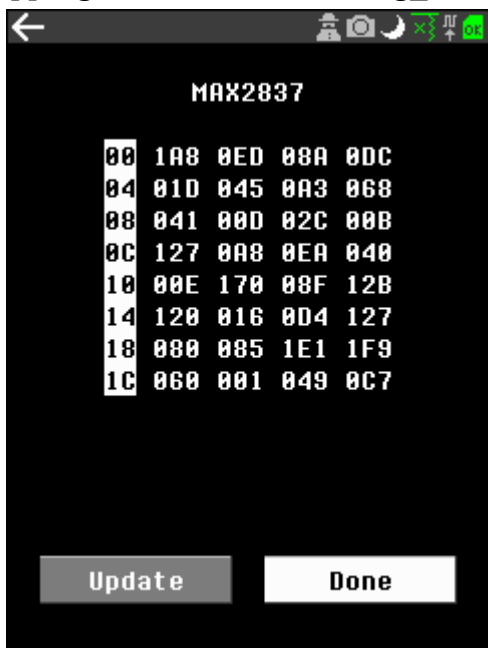
[[img/screenshots/Debug_Memory.png]]



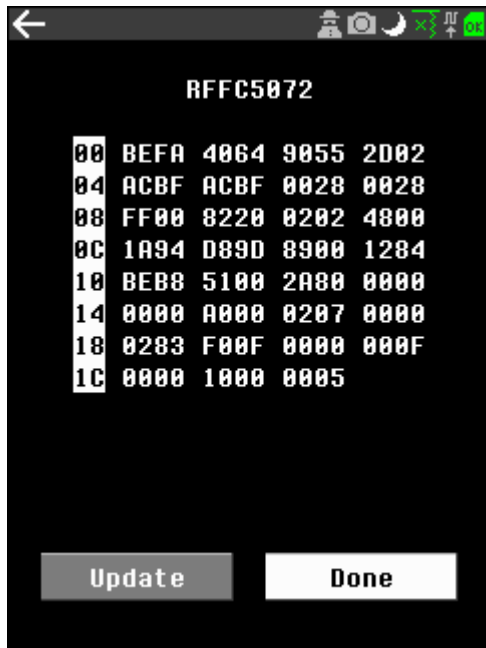
[[img/screenshots/Debug_Peripherals AK4951.png]]



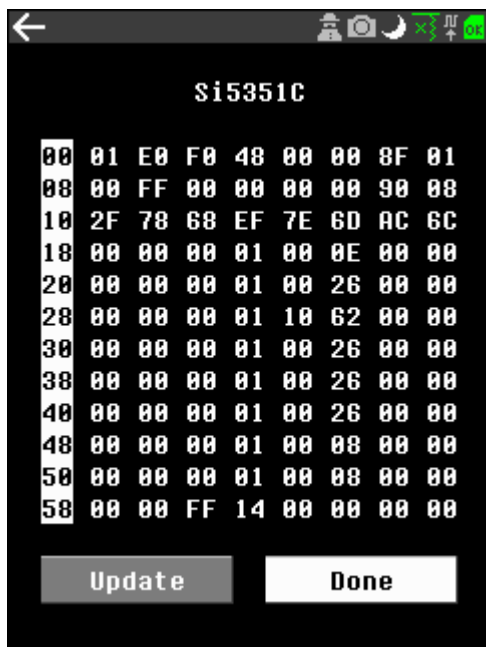
[[img/screenshots/Debug_Peripherals MAX2837.png]]



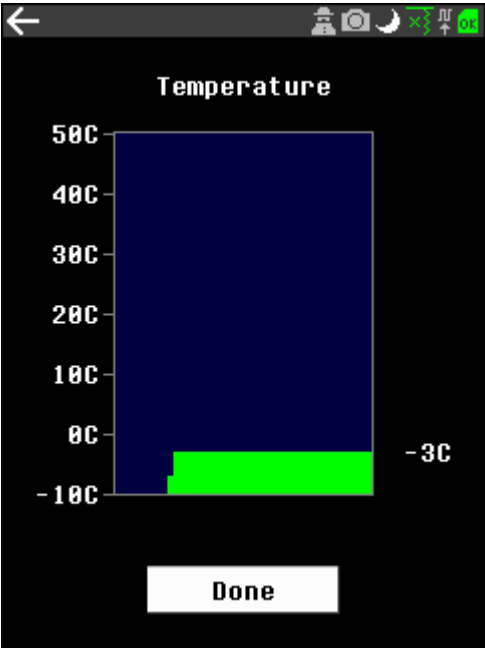
[[img/screenshots/Debug_Peripherals RFFC5072.png]]



[[img/screenshots/Debug_Peripherals Si5351C.png]]



[[img/screenshots/Debug_Peripherals Temperature.png]]



[[img/screenshots/Debug_Peripherals.png]]

The figure is a screenshot of a mobile application titled "Peripherals". It displays a list of four device IDs in a 2x2 grid. Each entry consists of a small icon and the device ID. The device IDs are RFFC5072, MAX2837, Si5351C, and AK4951. Below the grid is a large black rectangular area.

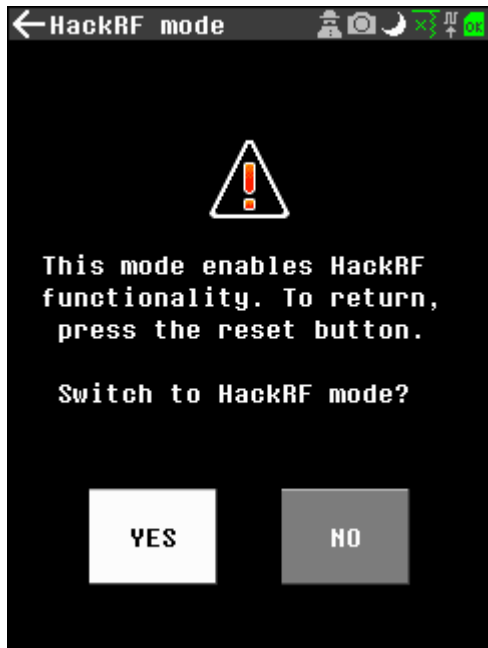
RFFC5072	MAX2837
Si5351C	AK4951

[[img/screenshots/Debug_SD Card.png]]

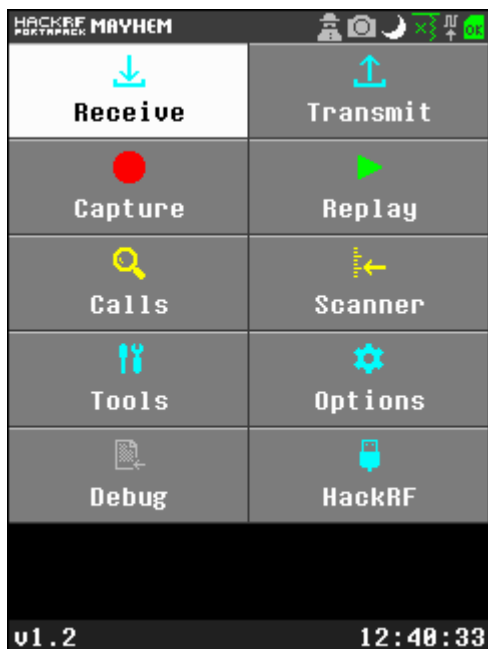
The figure is a screenshot of a mobile application titled "SD Card". It displays various details about an SD card. The details are organized into two main sections. The first section shows the card's CSD (Card Specific Data) and other basic information. The second section shows performance metrics. At the bottom, there are two buttons: "Test" and "OK".

SD Card	
CSD	400E0032 5B590000
	76B27F80 0A404013
Bus width	4
Card type	SD V2.0, SDHC
Block size	512
Block count	31116288
Capacity	15.931 GB
W ms	
W MB/s	
R ms	
R MB/s	
Test OK	

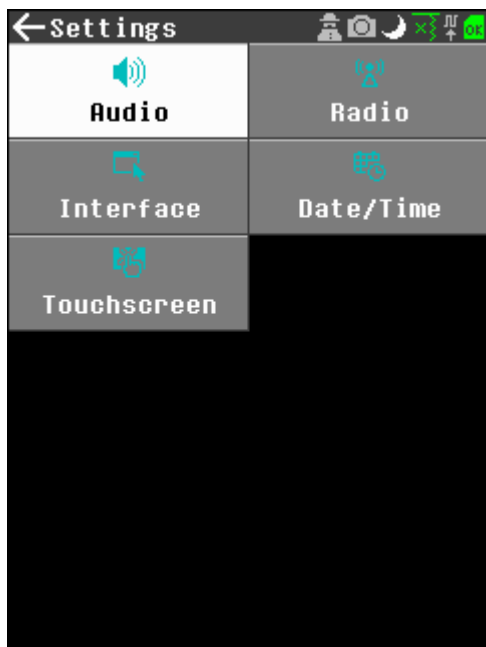
[[img/screenshots/HackRF.png]]



[[img/screenshots/Main.png]]



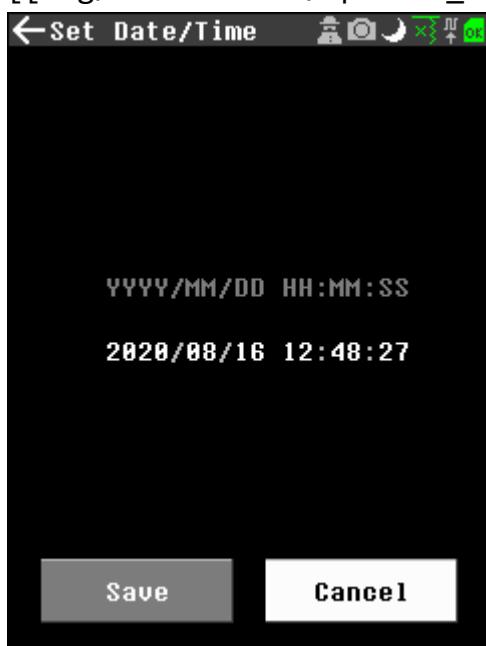
[[img/screenshots/Options.png]]



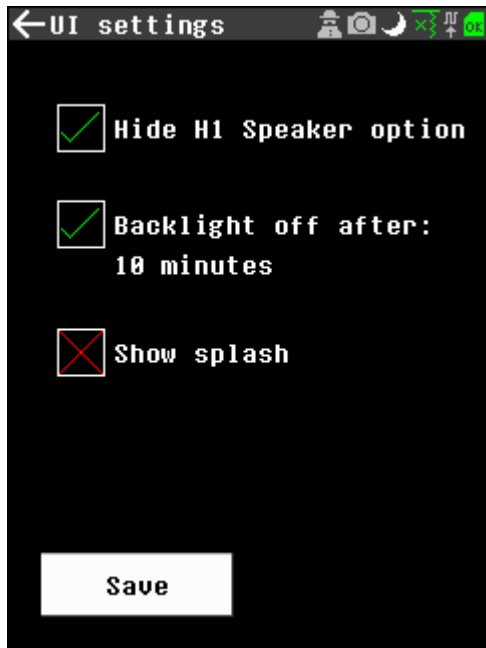
[[img/screenshots/Options_Audio.png]]



[[img/screenshots/Options_Date_Time.png]]



[[img/screenshots/Options_Interface.png]]



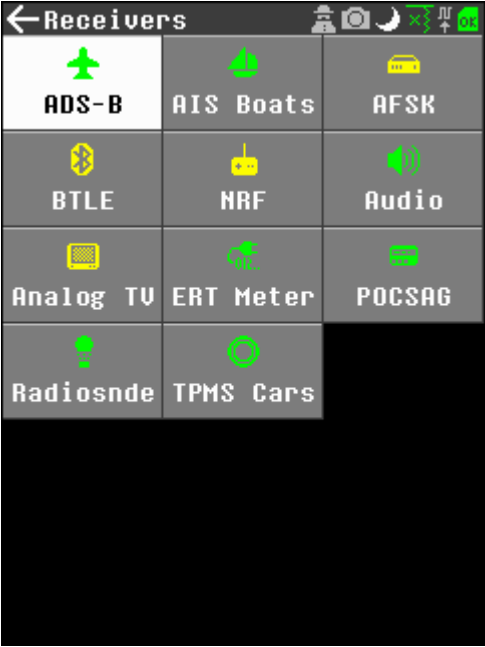
[[img/screenshots/Options_Radio.png]]



[[img/screenshots/Options_Touchscreen.png]]



[[img/screenshots/Receive.png]]



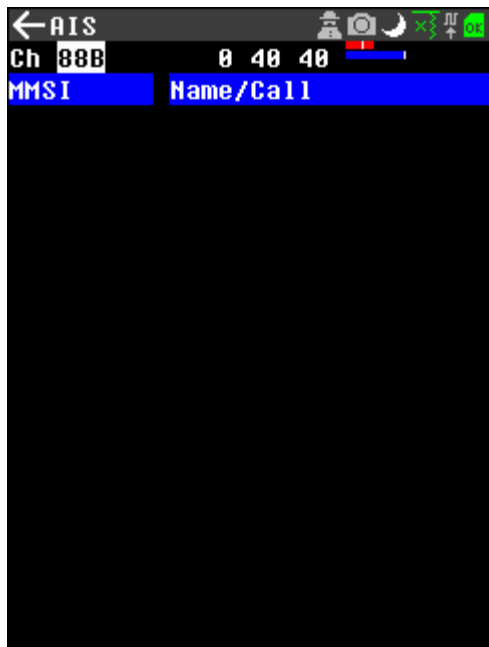
[[img/screenshots/Receive_ADS-B.png]]



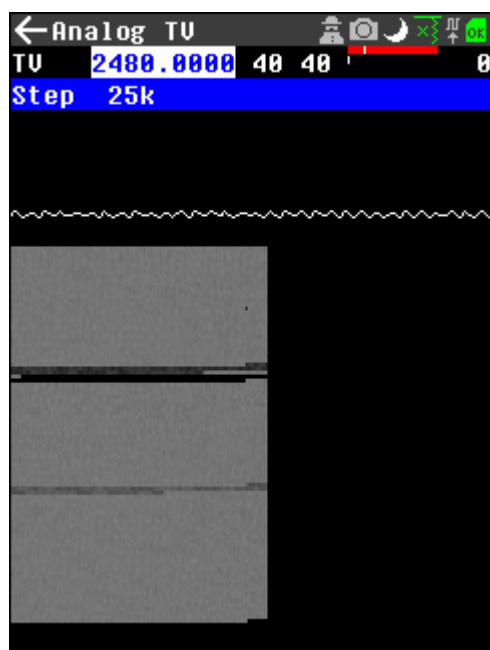
[[img/screenshots/Receive_AFSK.png]]



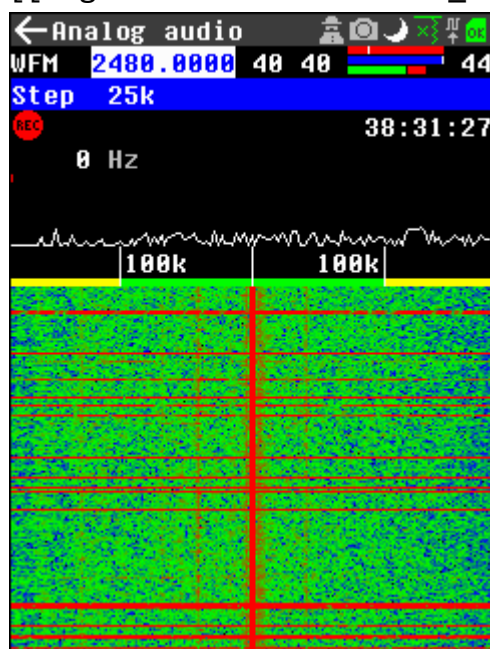
[[img/screenshots/Receive_AIS Boats.png]]



[[img/screenshots/Receive_Analog TV.png]]



[[img/screenshots/Receive_Audio.png]]



[[img/screenshots/Receive_BTLE.png]]



[[img/screenshots/Receive_ERT Meter.png]]



[[img/screenshots/Receive_NRF.png]]



[[img/screenshots/Receive_POCSAG.png]]



[[img/screenshots/Receive_Radiosnd.png]]



[[img/screenshots/Receive_TPMS Cars.png]]



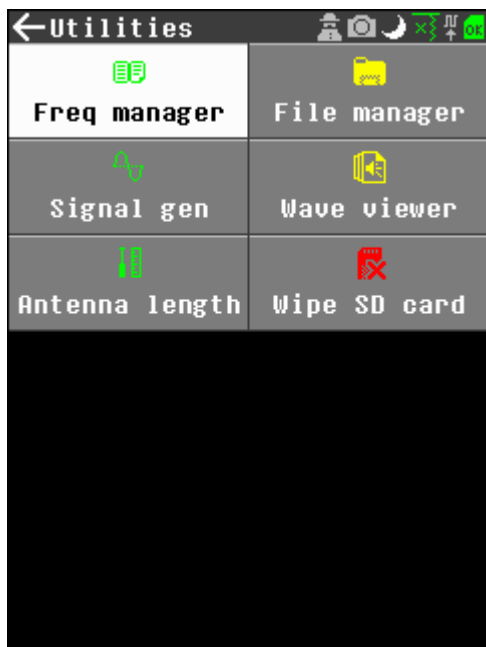
[[img/screenshots/Replay.png]]



[[img/screenshots/Scanner.png]]



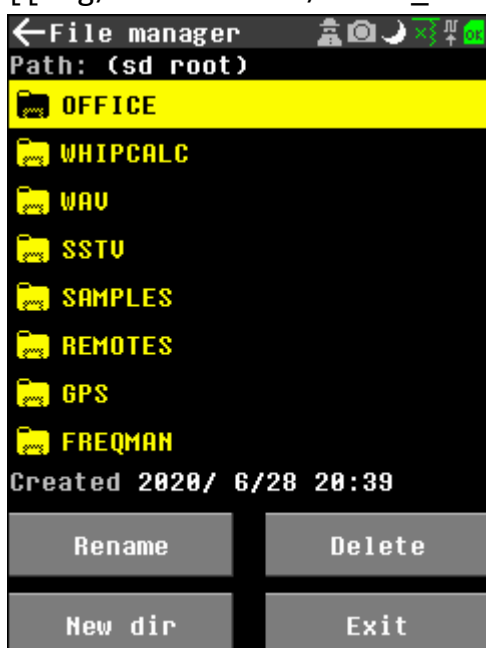
[[img/screenshots/Tools.png]]



[[img/screenshots/Tools_Antenna length.png]]



[[img/screenshots/Tools_File manager.png]]



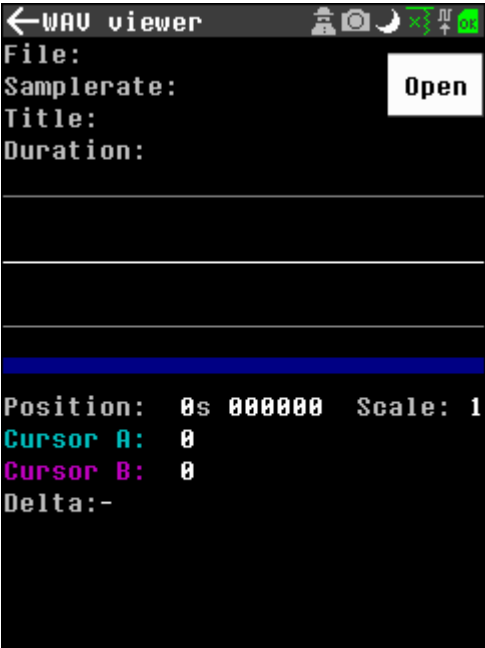
[[img/screenshots/Tools_Freq manager.png]]



[[img/screenshots/Tools_Signal gen.png]]



[[img/screenshots/Tools_Wave viewer.png]]



[[img/screenshots/Tools_Wipe SD card.png]]



[[img/screenshots/Transmit.png]]



[[img/screenshots/Transmit_ADS-B.png]]



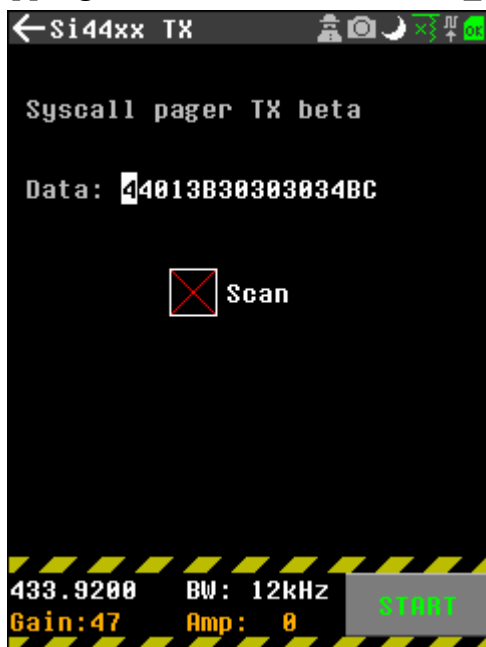
[[img/screenshots/Transmit_APRS.png]]



[[img/screenshots/Transmit_BHT Xy_EP.png]]



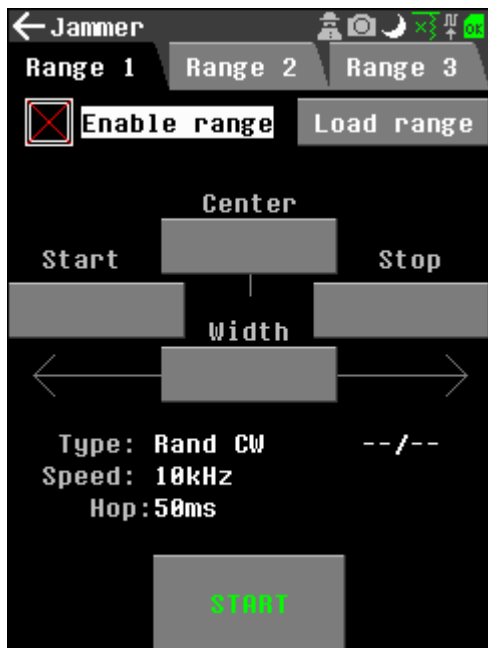
[[img/screenshots/Transmit_BurgerPgr.png]]



[[img/screenshots/Transmit_GPS Sim.png]]



[[img/screenshots/Transmit_Jammer.png]]



[[img/screenshots/Transmit_Key_fob.png]]



[[img/screenshots/Transmit_LGE_tool.png]]



[[img/screenshots/Transmit_Mic.png]]



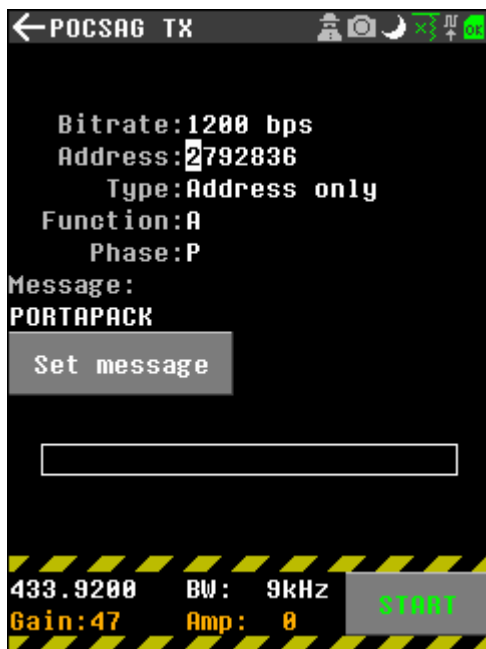
[[img/screenshots/Transmit_Morse.png]]



[[img/screenshots/Transmit_OOK.png]]



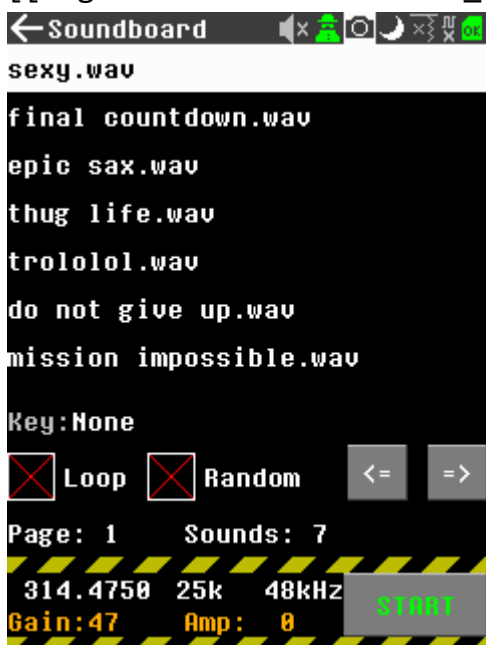
[[img/screenshots/Transmit_POCSAG.png]]



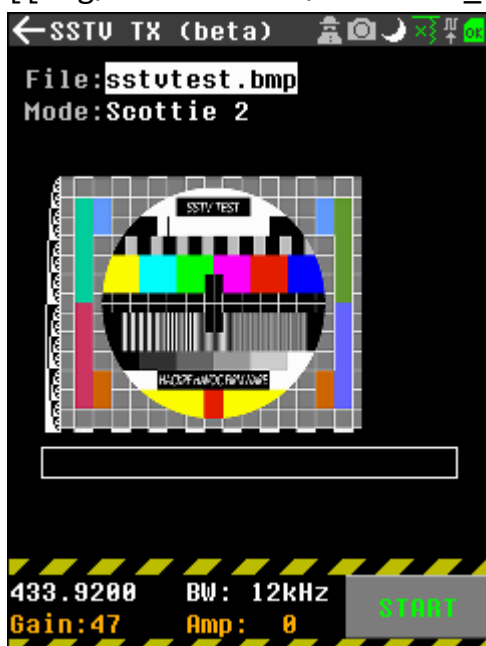
[[img/screenshots/Transmit_RDS.png]]



[[img/screenshots/Transmit_Soundbrd.png]]



[[img/screenshots/Transmit_SSTV.png]]



[[img/screenshots/Transmit_TEDI_LCR.png]]



Maintaining rules

- Once release created, don't edit it even if there's bug exist. Bug is normal on nightly, so don't panic.
- You shouldn't directly push to upstream, AKA this repo.
- Force push is not allowed in any case.
- At least one maintainer approve is needed for each PR (except the PR author) before merging.

Work Needed (Future Enhancement Suggestions)

Mayhem firmware is continually evolving. If you have the time and skill, there are several features that you could work on to contribute, including those listed below.

NOTE:

- To keep the list of Issues in github clean, some Enhancement suggestions have been Closed in github and moved to this list, so please include "closed" issues when searching github more information on any of the suggestions below.
- Due to ROM size limitations, new applications may need to be "external", meaning that they are loaded from a file on the SD card versus from the ROM.

Enhancement Wish List (and many more...)

Analog TV App

- Add support for NTSC.

Audio

- Add support for recording from Microphone to a WAV file.

Capture Files

- Add support for more advanced editing of capture files.

ERT Meters

- Add support for more utility meter types.

External Apps

- Move incomplete or infrequently-used apps external.

Port from Other Projects

- Support for additional RF devices/features may be ported to Mayhem from other open-source projects including rtl_433, Flipper Zero, etc.

SD Card

- Improve SD card performance.

Squelch

- Support squelch in all receiver modes.
- Standardize squelch meaning across apps.

TPMS App

- Add support for more types of TPMS sensors.

External App Roadmap


Untitled

Overview

More apps need to be moved from internal to external to make room for new features in the flash ROM. Considerations regarding exactly *which* apps should be moved from internal to external include the following:

1. Some apps should remain internal for better user experience (screens with fewer menu icons, and warnings about missing apps, are not the best look at first power-on).
2. Some core apps should remain internal for functional reasons, including the capability to install the external apps on the SD card (including the Flash Utility, SD-over-USB, and some Settings apps).
3. Some apps require the SD card to be installed and often to contain specific data files (such as map data or C8/C16 files) so it may make sense to move the app itself to the SD card too; no loss of functionality.
4. External app code is loaded into RAM (like the baseband M4 code), so will have less free RAM space available to operate, so apps that require more memory (including those that allocate lots of memory or use File Manager to select files) should *not* be moved to external.
5. Running code from RAM, external apps may have slightly better performance than internal apps that run from ROM (although they will still make many calls into the ROM code).
6. We also need to devise a way to sort the apps on the screen by the user and for consistency, since the external apps are currently at the end of the list and their position in the menu is *randomly* determined by their position in the FAT directory.

Legend

: Fully

✓: Partial

⊘: Not at all

Apps Currently in SPI Flash

(THE TABLES BELOW ARE VERY INCOMPLETE)

Catalog	App Name	Can work without sdcard?	Can work without correctly put sdcard content?	Importance	Current Suggest
Home	Capture	⊘	✓	✓ Core app	Stay in SPI flash
/	Replay	⊘	✓	✓ Core app	Stay in SPI flash
/	Remote	⊘	✓		
/	Scanner	✓	✓		
/	Microphone	✓	✓	✓ Core app	Stay in SPI flash
/	Looking Glass	✓	✓	✓ Core app	Stay in SPI flash
Receive	ADS-B	✓	✓		
/	AIS Boats	✓	✓		
/	APRS	✓	✓		
/	Audio	✓	✓	✓ Core app	Stay in SPI flash
/	BLE Rx	✓	✓		
/	ERT Meter	✓	✓		
/	Level	✓	✓		
/	POCSAG	✓	✓		
/	Radiosnde	✓	✓		
/	Recon	✓	✓		
/	Search				
/	TPMS	✓	✓		
/	Weather	✓	✓		
/	SubGhzD	✓	✓		

Apps Currently in sdcard

Catalog	App Name	Can work without sdcard if moved to SPI flash?	Can work without correctly put sdcard content if moved to SPI flash?	Importance	Current Suggest
Receive	AFSK				Stay in sdcard
Utilities	Pac-Man	✓	✓	Game	Stay in sdcard
Transmit	GPSSim	⊘	⊘		Stay in sdcard