Building a Custom Linux Image for a Raspberry Pi 4

Max-Felix Müller

2021
February

Contents

1 Project Goal

The goal for the project is to build a custom Linux image for a Raspberry Pi 4. Buildroot is used since it is free and getting familiar with it is necessary for further projects, as it will later be used to build a Linux image for custom hardware. Since building for a Raspberry Pi should be quite simple (one command?), there is the additional goal of modifying it. The final image should fulfill the following requirements:

- UART on the 40 pin GPIO header

2 Buildroot

Buildroot is a build system to build a root filesystem for Embedded Linux. It can build a toolchain, root filesystem, bootloader and kernel.

Buildroot is installed by cloning the directory structure from github.

```
git clone https://github.com/buildroot/buildroot.git buildroot
```

Make sure that the path before Buildroot does not include spaces in the filename.

The directories hold files needed for the build. The "board" directory has mappsing and configuration for the target board, such as the flash memory address and device tree files. In "configs" packages and properties can be added to the image. "packages" includes official packages available for Buildroot. Buildroot also allows for custom packages.

3 Building the Standard Linux Image

Since the Raspberry Pi is very good supported in the maker community, there are preconfigured files for it. To start off, after cloning buildroot, first prepare Buildroot and then select the configuration for the Raspberry Pi 4. Inside the buildroot directory:

```
sudo make clean
sudo make raspberrypi4_defconfig
```

The defconfig can be found in the "configs" directory of Buildroot.

After this is done, building the standard image starts with:

```
sudo make
```

During building a problem with libfakeroot occured. It could be fixed using the following patch for the file "/buildroot/output/host/libfakeroot.../libfakeroot.c

```
diff --git a/libfakeroot.c b/libfakeroot.c
index 3e80e38..14e56bc 100644
--- a/libfakeroot.c
+++ b/libfakeroot.c
@@ -90,6 +90,10 @@
 #define SEND_GET_XATTR64(a,b,c) send_get_xattr64(a,b)
 #endif
```

```
+#ifndef _STAT_VER
+#define _STAT_VER 0
+#endif
+
```

The standard build took 45 minutes for the first time. It uses all cores, so better CPUs with many cores speed up build time. Afterwards, the output is a image file that can be burnt on a micro SD card.

When inserting it in the Raspberry, the output was a command line Linux via UART. This was surprising, because it already satisfied the requirements for the project, but then again it was expected, since this is only a minimal Linux without any HDMI drivers. Booting took about 16 seconds and was mostly slowed down by trying to connect to a network which was not attached.

4 Understanding and Modifying the Standard Linux Image

Since the output was already a command line interface, the focus shifted more towards experimenting with different options in buildroot.

4.1 Setting a root Password

When logging in, it became obvious that the root user has no password set. After looking through menuconfig, an option was found to simply enter a password. The option can be found under system configuration.

The new build took less than a minute, since there are nearly no changes and most importantly no new packages to compile. It also worked just as before, with the additional step of entering the root password after logging in.

4.2 Adding a Package

Menucanfig can also be used to add a package. For example adding htop, a task manager, works by enabling the package in menuconfig > Target Packages > System Tools. The configuration has then to be saved and the image rebuild.

Whilst the command "htop" was previously not found, now it is. However there is another error. "btime" can not be found. Investigating further, it seems that the hardware clock is not set or not available.

A very important thing to know is that hex 0x03 is the character to send to abort an instruction. It is sent when using ctrl + c.

For now, the htop option is unselected again.

4.3 Research Online

After doing some research online, an introductary video was found on using buildroot to build an image for the Raspberry Pi. The video is for version 3, but the same rules should still apply.

- Change the C library to glibc. This is the GNU C library.
- Change the system hostname. The hostname is the name of the system.

- Change the system banner. It is shown before login.
- Enable timezones. Timezones are necessary to have automatic time updates.
- Show packages that are provided by BusyBox This will show additional packages to add.
- Select bash as the default shell. Bash only becomes available after selecting the previous option to show packages provided by BusyBox.

At this point, the next version of the image was buit. But somehow glibc threw an error when compiling so the C library was reverted to use uClibc-ng.

4.4 Language Support

To test language interpreters, Python3 is selected. From Libraries > Hardware handling > wiringpi might be interesting as it gives access to the GPIO header.

A separate build just for this option was not done, instead it was combined with the next big chage...

4.5 Enabling Graphics

Graphics requires firstly a window system. The main thing today is Xorg, but Wayland is making its way on becoming more popular.

To build Xorg the glibc C library is required. With Xorg, also a X server is required, as well as some libraries, but those are then enabled by default. Xinit is required to start the X server on startup.

As far as drivers go, keyboard and mouse are must have, as are fbdev and fbturbo. For joystick input, also select that one. Use the synaptics driver for touchpads.

Xterm is the default terminal for Xorg. You might want to install pcmanfm as a file manager and openbox as a very basic windows-like window manager. If you need a text editor other than Vi (if Vi means nothing to you definitly do this), also add leafpad. Another option would be Vim.

Screen, sudo, time and which are recommended as well. Propably time was the missing package earlier to handle htop properly, so that is also selected.

Sound is not used for now. It would propably require the alsa-utils and the alsa gui if needed. A browser is also missing, but depending on which browser, this would also require a 3d video driver like openGL.

Since there was a lot of packages added, the block size of the ext4 file system has to be increased. There are websites that allow the conversion of Gb to block size. For now, use the size 2097152, which corresponds to 1 Gb. Alternatively, since "120M" also seems to work, propably "1G" would also be fine.

Building this will take quite a while once again, since all the graphics packages are a good bit to download and build. But then the problem with glibc reappeared. Changing the gcc version seemed to help, but another error soon appeared.

4.6 Back to the Roots

Since there were quite a few changes made, the best option was to clean the build and restart. Python3 was added and is now working.

5 Adapting

It was possible to adapt the learned process of creating an embedded Linux image to build one for the EBAZ4205. That is a FPGA board originally used to mine cryptocurrencies, but now it can be bought for cheap from China.

The configuration files specific for the FPGA and the board layout can be found on GitHub. A Uboot patch had to be applied by hand, but that was just removing a two lines. The problem comes with the updated version of gcc.

Another problem that occured was the Linux Headers not being able to be downloaded. The source urls were responing, but the download itself returned an error. One possible reason for this is that the urls are the wrong ones, because custom urls have not been set properly. A possible solution is to change the version of the Linux kernel and the Linux headers. This might not always be the best solution, if for example a specific Linux version has to be used.

6 Fixes that have no Place

Here are some fixes I encountered that I could not fit into a specific section:

- $FORCE_UNSAVE_CONFIGURE$ - I had to use this sometimes. The problem is that I can not run make without sudo and this command is necessary to allow that. To use it, add $FORCE_UNSAVE_CONFIGURE = 1$ as an option to (sudo) make.
- Buildroot builds its own cross compiler, which takes some time. Instead it is possible to select an external toolchain in the toolchain menu of make-config. This tells Buildroot to use an external pre-build cross compiler which, for example, is provided by ARM for its architectures. In a first test, this method saved about 30 minutes from the estimated time without an external compiler.
- $YYLLOC$ - There are two files that have to be modified if this error occurs. They are in output > build > uboot > scripts > dtc. In the files dtc-lexer.l and dtc-lexer.lex.c comment or delete the lines "YYLTYPE yylloc;". This had to be done a second time in output > build > linux4.19 > scripts > dtc.

7 Final Thoughts

I am not sure how useful this documentation will be for other people, even for me. But the things I learned doing this "project" definitly brought me ahead on my quest to building my own embedded Linux system.

8 Links

$YYLLOC$
$https://github.com/Tomoms/android_kernel_oppo_msm8974/commit/11647f99b4de6bc460e106e876f72fc7a$