# Building Embedded Operating System with IMGUI Demo for *Raspberry π - 4 - model B* with *Yocto*

Kaloyan Krastev*

October 20, 2023

*Triple Helix Consulting[6]

# table of contents

# 1 introduction

These instructions[3] follow the configuration and build of a Linux-based operating system for *Raspberry* $\pi$ - 4 - *model B*[5] with *Yocto*[2]. Find project overview in [4].

The *operating system* (OS) build is done in several steps organized in corresponding sections as follows. Read in Section 2 how to fetch *metadata*. Section 3 shows how to configure the OS build. In Section 4 learn how to build the OS *image* and see how to copy *image* to *SD* card in Section 5. Section 6 is dedicated to post-install issues like the configuration of the WiFi interface from the command line.

# 2 metadata

*Metadata* is a set of instructions to build targets. It is organized in *recipe* files with the *.bb* suffix. Further there are *class* files with the suffix *.bbclass* with information shared between *recipes*. Finally, there are configuration files with the extension *.conf*. These define configuration variables to control the build process. *Metadata* is organized in *layers*. Layers logically separate information of a project. *OpenEmbedded*[1] defines the following layer types.

- base layers contain base *metadata* for the build
- machine aka *board support package* (BSP) layers include *hardware* (HW) support
- distribution layers hold the policy configuration
- *software* (SW) layers are used for additional SW
- miscellaneous layers do not fall in upper categories

The complete list of *github* SW *metadata* repositories used in this project includes *Yocto* layers, the *Raspberry* $\pi$ - 4 - *model B* BSP layer, a SW layer with custom recipes, and the build configuration itself. Please refer [4] for details.

In short, users fetch *metadata* in contrast to the *real data* fetched by *bitbake* during OS build. See Section 4 for details. It is an user decision where to put fetched *metadata*. However, it is nice to have all layer sub-directories in one location. In these instructions this location is referred as <layer_directory>. Execute next script to fetch layers from public *github* repositories. This shell script takes <layer_directory> as argument. You may download metafetch.sh here.

```
#!/usr/bin/env sh
# metafetch.sh
# fetch rpi metadata
error() { echo $1; exit 111; }
if [ ! "$#" -eq 1 ]; then error "usage: $0 <directory>";
if [ ! -d $1 ]; then error "error: $1 not a directory"; f

echo fetching metadata in $1 ...

exit 1

git clone -b kirkstone \
    git@github.com:yoctoproject/poky.git \
    $1/poky
git clone -b kirkstone \
    git@github.com:openembedded/meta-openembedded.git \
    $1/oe
git clone -b kirkstone \
    git@github.com:agherzan/meta-raspberrypi \
    $1/rpi/meta-raspberrypi
git clone \
    git@github.com:kaloyanski/meta-thc.git \
    $1/thc/meta-thc

exit 0
```

The second directory to create is the <build_directory> and I suggest that this one is not inside the <layer_directory> to not mix *data* and *metadata*. Fetch the project build configuration with the command that follows.

```
git clone git@github.com:TripleHelixConsulting/rpiconf.
```

<build_directory>/conf

# 3 configuration

After the last command from the previous section there should be two files in <build_directory>/conf, namely *local.conf* and *bblayers.conf*.

The path to *Yocto* layers is specified in *bblayers.conf*. Layer locations are wrong because most probably your <layer_directory> is not */home/yocto/layer*. Change this to correspond layers system path.

The build configuration is in *local.conf*. This should work as it is. Variables in this file control the build. I call them directives to avoid repetitions. Many directives are not covered in these instructions. Please refer *bitbake* documentation for details. It is not always easy to understand the meaning and the relations between different directives. What is more, *bitbake* syntax is pretty complicated. In short, your life can easily become unbearable if the build configuration is too long.

## 3.1 MACHINE

No doubt, this is the most important directive, set here to *raspberrypi*4-64. You may want to change this value if you build an OS for a different HW. If you want to examine OS built for *Raspberry* $\pi$ - 4 - *model B* on your host machine with *qemu*, set $MACHINE$ to *qemuarm*64. I confirm that this works although I did not find this approach very useful to test a *graphical user interface* (GUI).

## 3.2 PACKAGE_INSTALL

This is where to specify additional SW packages. This is useful for packages not included in the *image* by default. In my experience, the default OS has all necessary programs or compact alternatives. However this is the directive used to append *imgui*.

## 3.3 IMAGE_FSTYPES

This is another important directive. Here I have removed archived *images* that I do not need to decrease the built time and added the *wic* format to have an *image* file ready to be copied to the *SD* card immediately after the build. See Section 5 for details.

# 4 build

*Yocto* provides a list of image types. For obvious reasons, I have chosen $core-image-x11$[2] - a very basic X11 image with a terminal.

The primary build tool of *OpenEmbedded* based projects, such as the *Yocto* project, *bitbake*, works in the <build_directory>. Here is a list of the most important sub-directory names by default. These are configurable but usually there is no need to change their default names.

- <build_directory>/conf - build (*local.conf*) and layer (*bblayers.conf*) configuration files
- <build_directory>/downloads - fetched source code archives
- <build_directory>/tmp/work - working directory where source code is extracted, configured, compiled and installed
- <build_directory>/tmp/deploy/ipk - final SW packages in *ipk* format
- <build_directory>/tmp/deploy/images/raspberrypi4-64 - boot files, compiled kernels and OS *images*.

First, you need to initialize build environment.

```
source <layer_directory>/poky oe-init-build-env <build_
```

This will change your system path to <build_directory>. You may run now next command to check the project layers.

```
bitbake-layers show-layers
```

If this is fine, the following command is going to build the OS *image*.

| task | description |
|---|---|
| do_fetch | fetch the source code |
| do_unpack | unpack the source code |
| do_patch | apply patches to the source |
| do_configure | source configuration |
| do_compile | compile the source code |
| do_install | copy files to the holding area |
| do_package | analyse holding area |
| do_package_write_ipk | create *ipk* package |
| do_package_qa | quality checks on the package |

Table 1: A list of *bitbake* tasks

```
bitbake core-image-x11
```

Be patient because, unless your host machine is a supercomputer, this will take hours. Find a list of tasks performed by *bitbake* for a typical SW package in Table 1.

# 5 install

The OS includes a kernel $ARM$, 64 *bit* boot executable *image* of $23MB$, a $Raspberry\ \pi$ - 4 - $model\ B$ configuration of Linux 5.15. The total size of kernel modules is $21MB$. Happily this kernel release has a $long-term\ support$ (LTS).

*Yocto* provides multiple package and *image* formats. Further, different ways exist to install *images* on $SD$ card. The result is an OS with two partitions - */root* and */boot*. There are not *swap* and *home* partitions. I recommend the classic command-line tool *dd* to copy data. It works fine with different *image* formats like $rpi-sdimg$, *hddimg* and *wic*. The last format is recommended. Find the card

device name, usually */dev/sda*, unmount it with *umount* if mounted, and do copy data with a simple command

*dd if=core-image-x11-raspberrypi4-64.wic of=/dev/sda status=progre*

note 1: run this command in <build_directory>/tmp/deploy/images/raspb
64
note 2: run this command with *root* privileges
note 3: be careful to not specify the device name of your hard drive (see note 2)

The transfer is going to take a while. Once it is over, put the card in you *Raspberry* $\pi$ - 4 - *model B* and turn it on. That's it.

# 6 run

Connected embedded systems can communicate to one another and to cloud-based *platform-as-a-service* (PaaS) solutions. In addition, a remote control may be required. An *secure shell* (SSH) server is a standard solution for both problems.

Wireless connection is established via classic command-line tools like *ip*, *iw*, *dhcpcd*, and *wpa_supplicant*. Custom shell scripts are installed in */usr/bin*, as well as a running GUI example to demonstrate the usage of the *Dear ImGui* library. Once an *internet protocol* (IP) address is assigned, the SSH server by *Dropbear* allows for a secured remote login, remote control and file transfer.

# 7 outlook

This reports the progress in the development of a custom Linux-based OS for *Raspberry* $\pi$ - 4 - *model* *B*[5]. The kernel version of this embedded OS is Linux release 5.15. An example GUI application using the *Dear ImGui* library is built as a part of the OS *image*. In addition, an SSH server provides remote connection, data transfer and device control. As the OS is now functional, performance and real-time tests are ongoing.

# acronyms

**BSP** *board support package*

**SSH** *secure shell*

**GUI** *graphical user interface*

**SW** *software*

**HW** *hardware*

**OS** *operating system*

**IP** *internet protocol*

**PaaS** *platform-as-a-service*

**LTS** *long – term support*

# bibliography


[1] Yocto Project community. *OpenEmbedded*. 2017. URL: https://www.openembedded.org (visited on 2023).

[2] Yocto Project community. *Yocto Project*. 2023. URL: https://www.yoctoproject.org (visited on 2023).

[3] Kaloyan Krastev. *Building Embedded Operating System with IMGUI Demo for Raspberry π - 4 - model B with* Yocto. 2023. URL: https://kaloyanski.github.io/meta-thc/thchowto.html (visited on 2023).

[4] Kaloyan Krastev. *Embedded Operating System for Raspberry π - 4 - model B with* Yocto. 2023. URL: https://kaloyanski.github.io/meta-thc/thcport.html (visited on 2023).

[5] Raspberry π Ltd. *Raspberi π*. 2023. URL: https://www.raspberrypi.com (visited on 2023).

[6] Atanas Rusev. *Triple Helix Consulting*. 2023. URL: https://triplehelix-consulting.com (visited on 2023).