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

Kaloyan Krastev*

October 18, 2023

*Triple Helix Consulting[9]

# table of contents

# 1 introduction

These instructions[5] follow the configuration and build of a Linux-based operating system for *Raspberry* $\pi$ - 4 - *model B*[7] with *Yocto*[2]. Find project overview in [6].

The *operating system* (OS) build is done in four steps and instructions are organized in four corresponding sections as follows.

- section 2 get *metadata*
- section 3 configure OS build
- section 4 build OS *image*
- section 5 copy *image* to *SD* card

Section 6 is dedicated to post-install issues like the configuration of the WiFi interface from the command line.

# 2 download

*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 layer
  base *metadata* for the build

- machine aka *board support package* (BSP) layer
  *hardware* (HW) support

- distribution layer
  policy configuration

- *software* (SW) layer
  additional SW

- miscellaneous layer
  for layers that 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 [6] for details.

It is up to users to decide where to download *metadata*. It is a good idea to have all layer sub-directories in one location. In these instructions this is referred as <layer-directory>. Once you have it created execute next lines to get layer *metadata*.

```
git clone -b kirkstone \
git@github.com:yoctoproject/poky.git \
<layer-directory>

git clone -b kirkstone \
git@github.com:openembedded/meta-openembedded.git \
<layer-directory>

git clone -b kirkstone \
git@github.com:agherzan/meta-raspberrypi \
<layer-directory>/rpi

git clone git@github.com:kaloyanski/meta-thc.git \
<layer-directory>/thc/meta-thc
```

The second directory to create is the <build-directory> and I strongly suggest that this one is not inside the <layer-directory>. Download build configuration in the <build-directory>.

```
git clone git@github.com:TripleHelixConsulting\
/yocto_x86_BasicConfig.git \
<build-directory>/conf
```

# 3 configuration

## 3.1 layers

Here is a list of *Yocto* layers. The project reference distribution is *poky*.

- $meta$
  User-space data

- $meta - poky$
  $Yocto$ reference distribution

- $meta - raspberrypi$
  This[3] is the general HW specific BSP overlay for the $RaspberryPi$ device. The core BSP part of $meta - raspberrypi$ works with different $OpenEmbedded/Yocto$ distributions and layer stacks. In short, the recipes to build the kernel and kernel modules are in this layer. For details see the package *linux-raspberrypi*. In addition, here is the HW specific firmware. By chance, the build configuration corresponds the specific HW, in this case $Raspberry$ $\pi$ - 4 - $model\ B$.

- $meta - thc$
  I have introduced a new $Yocto$ SW layer to control the build of $Dear\ ImGui$ and $GLFW$. As long as the source codes have a standard build configuration, the *bitbake* recipes are straightforward. Both instructions inherit *cmake*.

# 4 build

```
cd <layer-directory>/poky
source oe-init-build-env <build-directory>
bitbake core-image-x11
```

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 there is no need to change their default names.

- <build-directory>/conf - build (*local.conf*) and layer (*bblayers.conf*) configuration files
- <build-directory>/downloads - downloaded source code archives, usually fetched from github.com
- <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 - OS and boot images and compiled kernels

## 4.1 configuration

*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. In the main build configuration, apart from *Dear ImGui* and *GLFW*, I have added the following packages;

- $os-release$
  OS identification

- *Dropbear*
  Compact *secure shell* (SSH) server[4]

- *dhcpcd*
  *dynamic host configuration protocol* (DHCP) client[8]

- *thcp*
  OS post-configuration scripts

# 5 install

The total size of the operating system is between from 250 up to 384$MB$ or 79$MB$ *tar.bz* archive, including kernel $ARM$, 64 *bit* boot executable *image* of 23$MB$, a *Raspberry* $\pi$ - 4 - *model B* configuration of Linux 5.15. The total size of kernel modules is 21$MB$. Happily this kernel release has a $long-term$ *support* (LTS). The list of packages included in the OS *image* in Table 1 gives a good idea of the contents.

*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 only - $/root$ 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 it is mounted, and do copy data with the simple command

| package | description |
|---|---|
| packagegroup-core-boot | boot |
| packagegroup-base-extended | base |
| run-postinsts | post |
| opkg | package manager |
| psplash-raspberrypi | *Raspberry* $\pi$ - 4 - *model B* splash |
| packagegroup-core-x11-base | the *X* server |
| os-release | OS identifier |
| dropbear | SSH server |
| dhcpcd | DHCP client |
| thcp | SW layer |
| glfw | *OpenGL* |
| imgui | *Dear ImGui* |

Table 1: A list of packages in *core-image-x11-raspberrypi4-64*

*dd if=whatever.wic of=/dev/sda status=progress*

Run this command with *root* privileges and be careful to not specify the device name of your hard drive. This will take a while. When 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 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 *graphical user interface* (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*[7]. 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*

**DHCP** *dynamic host configuration protocol*

**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] Andrei Gherzan. *meta-raspberrypi*. 2023. URL: https://github.com/agherzan/meta-raspberrypi (visited on 2023).

[4] Matt Johnston. *Dropbear SSH*. 2023. URL: https://matt.ucc.asn.au/dropbear/dropbear.html (visited on 2023).

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

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

[7] Raspberry $\pi$ Ltd. *Raspberi* $\pi$. 2023. URL: https://www.raspberrypi.com (visited on 2023).

[8] Roy Marples. *dhcpcd*. 2023. URL: https://roy.marples.name/projects/dhcpcd (visited on 2023).

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

powered by LaTeX

this document in pdf format