

Embedded Operating System for *Raspberry π - 4 - model B* with *Yocto*

Kaloyan Krastev*

October 4, 2023

*Triple Helix Consulting[[11](#)]

table of contents

1	introduction	3
2	sources	5
2.1	application	5
2.2	layers	6
3	build	7
3.1	configuration	7
3.2	image	7
4	connection	8
5	outlook	9

1 introduction

This article[8] reports the progress in the configuration and build of a Linux-based operating system for *Raspberry π - 4 - model B*[9] with *Yocto*[3]. The *central processing unit* (CPU) is *ARM Cortex-A72*. It is a high performance CPU with low power consumption. With dimensions of 9 x 7 x 2 *cm*, the machine has 4 *GB* of RAM. As there are no fixed storage devices, *images* are installed on *SD* cards.

Embedded devices[1] are compact systems with specific purpose. Embedded operating systems provide a limited number of services defined by this purpose and device *hardware* (HW). There are important reasons why Linux is a preferred *operating system* (OS) kernel for embedded devices. Since such devices usually have limited HW resources, they rely on optimized *software* (SW) implementations. Smaller and faster than their original, such programs have limited, but usually sufficient functionality. For example, *BusyBox* combines tiny versions of many common *unix* utilities into a single executable[2]. Another example is *Dropbear SSH* - an optimized *secure shell* (SSH) server implementation[7]. *Dropbear* is significantly smaller in size compared to *OpenSSH*. Unlike GNU/Linux desktop distributions[12], embedded Linux has neither a *graphical user interface* (GUI) for system configuration nor a centralized service manager like *systemd*.

Yocto Project[3] provides a popular framework for configuration and build of Linux-based operating systems. First of all, *Yocto* supports HW via a SW layer called *board support layer* (BSP). In addition, there are custom distribution configuration and a tool called *Bitbake* to create SW packages and to build OS *images*. These *images* are configurable and operational. There is a bootloader, a kernel release and the user-space part of the OS, including custom user

applications. This approach is indispensable when it comes to embedded devices. *Yocto* images are collections of [SW](#) packages. Packages are created via [SW](#) recipes. Although a recipe may have all sorts of instructions, a typical one contains the source code location and the [SW](#) build configuration. These are architecture independent. However, binaries are cross-compiled in case of different target machine processor architecture. This applies to the entire [OS](#).

2 sources

As they differ, it could be extremely useful to isolate the [SW](#) development from the [OS](#) build. This way developers may work and test a [SW](#) application on their own. As far as I could fetch the source code, in example, from a *git* repository, in theory, it should not be too complicated to build an [OS](#) able to run this application. What is more, I can build it for a computing device of my choice. I just need the corresponding [BSP](#).

A complete list of *github* [SW](#) repositories used in this project includes *Yocto*, the [BSP](#), a [SW](#) layer with custom recipes, the configuration and the source code of the application and the dependencies. Note that for a relatively simple application I must fetch six [SW](#) repositories. Follow links for details.

- *Yocto* reference distribution yoctoproject.org/poky.git
- [BSP](#) layer for *Raspberry* π boards [agherzan/meta-raspberrypi.git](https://github.com/agherzan/meta-raspberrypi)
- *Yocto* configuration [TripleHelixConsulting/yocto_x86_BasicConfig.git](https://github.com/TripleHelixConsulting/yocto_x86_BasicConfig)
- [SW](#) layer [kaloyanski/meta-thc.git](https://github.com/kaloyanski/meta-thc)
- Immediate mode [GUI](#) [kaloyanski/imgui_aarch64_glfw_opengl2_experimental](https://github.com/kaloyanski/imgui_aarch64_glfw_opengl2_experimental)
- OpenGL library [glfw/glfw.git](https://github.com/glfw/glfw)

2.1 application

Dear ImGui[\[4\]](#) is a bloat-free [GUI](#) library for C++. It outputs optimized vertex buffers that you can render anytime in your 3D-pipeline-enabled application. It is fast, portable, renderer agnostic,

and self-contained (no external dependencies). *Dear ImGui* is designed to enable fast iterations and to empower programmers to create content creation tools and visualization/debug tools (as opposed to UI for the average end-user). It favors simplicity and productivity toward this goal and lacks certain features commonly found in more high-level libraries. *Dear ImGui* is particularly suited to integration in game engines (for tooling), real-time 3D applications, full-screen applications, embedded applications, or any applications on console platforms where operating system features are non-standard.

Dear ImGui depends on *GLFW*[\[5\]](#), an open-source, multi-platform library for *OpenGL*, *OpenGL ES* and *Vulkan* development on the desktop. It provides a simple API for creating windows, contexts and surfaces, receiving input and events. *GLFW* is written in *C* and supports *Windows*, *macOS*, *X11* and *Wayland*.

Dear ImGui is licensed under the *MIT* License. *GLFW* is licensed under the *zlib/libpng* license.

2.2 layers

Here is a list of layers added to the standard *Yocto* layers. The project reference distribution is *poky*.

- *meta – raspberrypi*

This[\[6\]](#) 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.

- *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*.

3 build

3.1 configuration

Yocto provides a list of image types. For obvious reasons, I have chosen *core-image-x11*[3] - 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 SSH server[7]
- *dhcpcd*
dynamic host configuration protocol (DHCP) client[10]
- *thcp*
OS post-configuration scripts

3.2 image

The total size of the operating system is 250MB or 80MB *tar.bz* archive, including kernel *ARM*, 64 bit boot executable *image* of 23MB, based on Linux 5.15. This is a kernel release with a long-term support. The list of packages included in the *image* in Table 1 gives a good idea of the OS contents.

Yocto provides multiple package and *image* formats. Further, different ways exist to install *images*. Finally, formats do not matter, as long as the result is a complete OS on an *SD* card. I recommend the classic tool *dd* to copy data.

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

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

4 connection

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 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.

5 outlook

This reports the progress in the development of a custom Linux-based OS for *Raspberry π - 4 - model B*[9]. This embedded OS is based on Linux kernel 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 layer*

SSH *secure shell*

GUI *graphical user interface*

SW *software*

HW *hardware*

OS *operating system*

DHCP *dynamic host configuration protocol*

CPU *central processing unit*

IP *internet protocol*

PaaS *platform-as-a-service*

bibliography

- [1] Frank Vasquez and Chris Simmonds. *Mastering Embedded Linux Programming*. Packt Publishing Limited, 2021.
- [2] Erik Andersen. *BusyBox*. 2023.
URL: <https://busybox.net> (visited on 2023).
- [3] Yocto Project community. *Yocto Project*. 2023.
URL: <https://www.yoctoproject.org> (visited on 2023).
- [4] Omar Cornut. *Dear ImGui*. 2023.
URL: <https://github.com/ocornut/imgui> (visited on 2023).
- [5] Marcus Geelnard and Camilla Löwy. *OpenGL*. 2023.
URL: <https://www.glfw.org> (visited on 2023).
- [6] Andrei Gherzan. *meta-raspberrypi*. 2023.
URL: <https://github.com/agherzan/meta-raspberrypi> (visited on 2023).
- [7] Matt Johnston. *Dropbear SSH*. 2023.
URL: <https://matt.ucc.asn.au/dropbear/dropbear.html> (visited on 2023).
- [8] Kaloyan Krastev. *Embedded Operating System for Raspberry π - 4 - model B with Yocto*. 2023.
URL: <https://kaloyanski.github.io/meta-thc> (visited on 2023).
- [9] Raspberry π Ltd. *Raspberi π* . 2023.
URL: <https://www.raspberrypi.com> (visited on 2023).
- [10] Roy Marples. *dhcpcd*. 2023.
URL: <https://roy.marples.name/projects/dhcpcd> (visited on 2023).

- [11] Atanas Rusev. *Triple Helix Consulting*. 2023. URL:
<https://triplehelix-consulting.com> (visited on 2023).
- [12] Atanas Georgiev Rusev. *Manjaro Linux User Guide*.
Packt Publishing Limited, 2023.