Oniro Project

Release 1.0.0-rc

Oniro Project

Dec 27, 2021

CONTENTS

Welcome to Oniro Project documentation!

You are welcome to take a tour and play with Oniro's initial code contribution. It is in the final stage toward becoming the project's official codebase. If you feel like joining, we would love to have you among the list of Oniro's initiating supporters. These are exciting times! There couldn't be a better moment for joining Oniro!

To learn more about the Oniro Project, go to https://oniroproject.org.

Note: Oniro™ is a registered trademark of Eclipse Foundation.

Oniro Project is an OpenHarmony compatible open-source project aimed at reducing fragmentation in the consumer and IoT device industry by providing a common technology that can power devices irregardless of their make or model. It currently provides the base of an ecosystem where partners can drive a unified platform aiming at IoT products. As the ecosystem evolves, the project will tackle a feature-proof distributed operating system as part of an all-scenario strategy initiative, adaptable to mobile devices, fitness and health targets, entertainment systems, and so on. Unlike a legacy operating system that targets a specific device, Oniro Project will adopt a distributed architecture design based on a set of system capabilities. Starting with a set of reference platforms, Oniro Project will be flexible enough to target a wide range of devices that accompany users' daily life.

CHAPTER

ONE

ONIRO PROJECT OVERVIEW

1.1 Oniro Project Vision and Aims

1.1.1 System Positioning

Oniro Project is an ambitious rethink on how an open source collaborative operating system can be run across a variety of device classes - from small microcontrollers with kilobytes of memory to powerful CPUs driving a phone, laptop or even a data center. The goal of Oniro Project is to evolve a set of builtin system capabilities on top of commodity open source kernels that allows sharing of resources and collaboration across distributed devices of various classes.

The development of these capabilities will happen through an open community project along with other interested parties.

- For an end-user, Oniro Project will integrate the multiple, standalone smart devices owned by the user and allow for fast interconnection, capability collaboration, and resource sharing between them. This way, the individual devices can collaborate to provide better context-aware services than if they were operating independently of each other.
- For an application developer, Oniro Project will integrate distributed technologies to ease application development across different device classes. Developers will be able to focus on upper-layer service logic and develop richer, collaborative applications more easily.
- For device developers, Oniro Project will provide reference software blueprints for the key product verticals that will allow them to focus their time on tailoring the OS to their device's resource capabilities and service characteristics. These software blueprints will provide best-in-class practices and solutions to keep the devices secure out-of-the-box.

Oniro Project will support APIs in multiple programming languages depending on the constraints of the underlying hardware. Developers will be able to choose from Java, Extensible Markup Language (XML), C/C++, JavaScript (JS), Cascading Style Sheets (CSS), and HarmonyOS Markup Language (HML) to develop applications for Oniro Project.

1.1.2 Technical Architecture

Oniro Project has a layered architecture built around the Yocto Project and bitbake build system. The Yocto Project is very popular in the embedded Linux community and provides an excellent platform for developing a highly-customizable, cross-kernel operating system. From bottom to top, Oniro Project consists of the kernel layer, system services layer, framework layer, and application layer. In multi-device development, Yocto provides the capabilities to tweak layers and recipes to remove unnecessary subsystems, functions, or modules as required.

Kernel Layer

System Services Layer

Framework Layer

Application Layer

Oniro Project will support a multi-kernel design out of the box (Linux kernel and an RTOS such as Zephyr RTOS or LiteOS) so that appropriate OS kernels can be selected for devices with different resource limitations. Over time, a kernel abstraction layer (KAL) will shield differences in kernel implementations and provide the upper layer with basic kernel capabilities, including process and thread management, memory management, file system, network management, and peripheral management.

The System Services Layer will contain the bulk of the differentiating features of Oniro Project. It will provide a complete set of capabilities essential for Oniro Project to offer services for applications through the framework layer. The system services layer will add the following features over time:

- The protocols and primitives that allow devices to discover each other
- APIs to allow sharing of computing, storage and other resources
- APIs that allow applications to be more context-aware due to collaboration with other devices in the network
- APIs to allow applications to expose business logic as *abilities* that may be integrated into other applications or even used on other devices in the network

The Framework layer will provide an SDK to develop Oniro Project applications in multiple languages such as Java, C, C++, and JS depending on the target device class and its HW constraints.

When completed, the Application layer will host the system and third-party applications. Oniro Project applications will be able to use APIs to expose business logic as *abilities* that may be utilized inside other applications, thus allowing creation of more integrated experiences on the same device as well as distributed across devices.

CHAPTER

TWO

BUILD SYSTEM GUIDE

2.1 Oniro Project - Quick Build

This section will guide you to building your first Oniro Project image targeting a supported reference hardware. It will also provide the steps for flashing and booting such an image.

The steps below will focus on a Qemu-based target. If you want to get a feeling of Oniro Project on a real hardware, checkout the *Avenger96 support page*.

Contents

2.1.1 Prerequisites

Install all the required host packages. Here is an example for **Ubuntu**:

```
$ sudo apt-get install gawk wget git diffstat unzip texinfo gcc-multilib \
  build-essential chrpath socat cpio python3 python3-pip python3-pexpect \
  xz-utils debianutils iputils-ping python3-git python3-jinja2 libegl1-mesa libsdl1.2-
↪dev \
  pylint3 xterm
```

See official Yocto documentation for host package requirements on all supported Linux distributions.

2.1.2 Clone Build System Repositories

Install Google git repo tool. For example, on **Ubuntu 20.04**, you can do this by:

```
$ sudo add-apt-repository ppa:ostc/ppa
$ sudo apt-get update
$ sudo apt-get install git-repo
```

Initialize a repo workspace and clone all required repositories:

```
$ mkdir oniroproject; cd oniroproject
$ repo init -u https://booting.oniroproject.org/distro/oniro
$ repo sync --no-clone-bundle
```

2.1.3 Building an Oniro image

The following steps will build a `oniro-image-base`. The process will build all its components, including the toolchain, from source.

First of all change directory into the one where the build repositories were cloned using the repo tool. See above.

Note: Depending on the configuration type, a single Oniro Project build could use around 100GB of disk space for downloads, temporary files, and build artifacts combined.

Initialize the build directory and run a build:

```
$ TEMPLATECONF=../oniro/flavours/linux . ./oe-core/oe-init-build-env build-oniro-linux
$ MACHINE=qemux86-64 bitbake oniro-image-base
```

2.1.4 Booting a Qemu X86-64 Target with a Oniro Project image

Once the image is built, you can run a Qemu X86-64 instance using the provided script wrapper as follows:

```
$ MACHINE=qemux86-64 runqemu oniro-image-base wic
```

If the host has a VT-capable CPU, you can pass the `kvm` argument for better performance. Check `runqemu`'s help message for all available arguments.

2.2 Oniro Project, a Yocto-based Build System

Oniro Project build system, the foundation of the build infrastructure, is based on Poky, the Yocto Project open source reference embedded distribution. This section details both generic and Oniro Project specific aspects of the build system.

2.2.1 Poky/Yocto Project

Oniro Project aims to use standard opensource tools to create a build environment that is both familiar to users in the domain but also flexible enough for the requirements of the project. With this in mind, the project build infrastructure is based on the OpenEmbedded build system, more specifically Poky, the Yocto Project open source reference embedded distribution.

Contents

Build System Concepts

The build system uses build instruction files that in the language of the system are called `recipes` and `layers`.

Layers are one of the fundamental models of the build system. It enables both collaboration and customization by defining scoped meta-data. These layers become a collection of build instruction files that have a defined scope. For example, there are BSP (board support package) layers that enable board support in the build system.

See terms for reference for more information.

Oniro Project Build Layers

Oniro Project bases its build setup on OE-core and bitbake. The main hub of layers, is oniro, a collection of layers with different scopes for defining the project's requirements and capabilities.

For example, `meta-oniro-core` provides build recipes for defining the core policies of the build infrastructure (*distribution* configuration, images, core packages customization, etc.).

Another example is `meta-oniro-staging`, a layer that provides temporary fixes and support for changes that are aimed upstream but have this place until upstream catches up.

For more details of each provided layer of `oniro`, see the relevant `README.md` file at the root of the layer.

Besides the `oniro` collection of layers, the project is also the home to a set of other build system layers. Explore them all in our project GitLab instance.

Additional Documentation

Yocto Project provides extensive documentation on various aspects of the build system. For the general usage of the build system, it's components, architecture and capabilities consult the following resources:

- Yocto Project Documentation Home
- Yocto Project Quick Build
- Yocto Project Reference Manual

2.2.2 Oniro Project Build Architecture

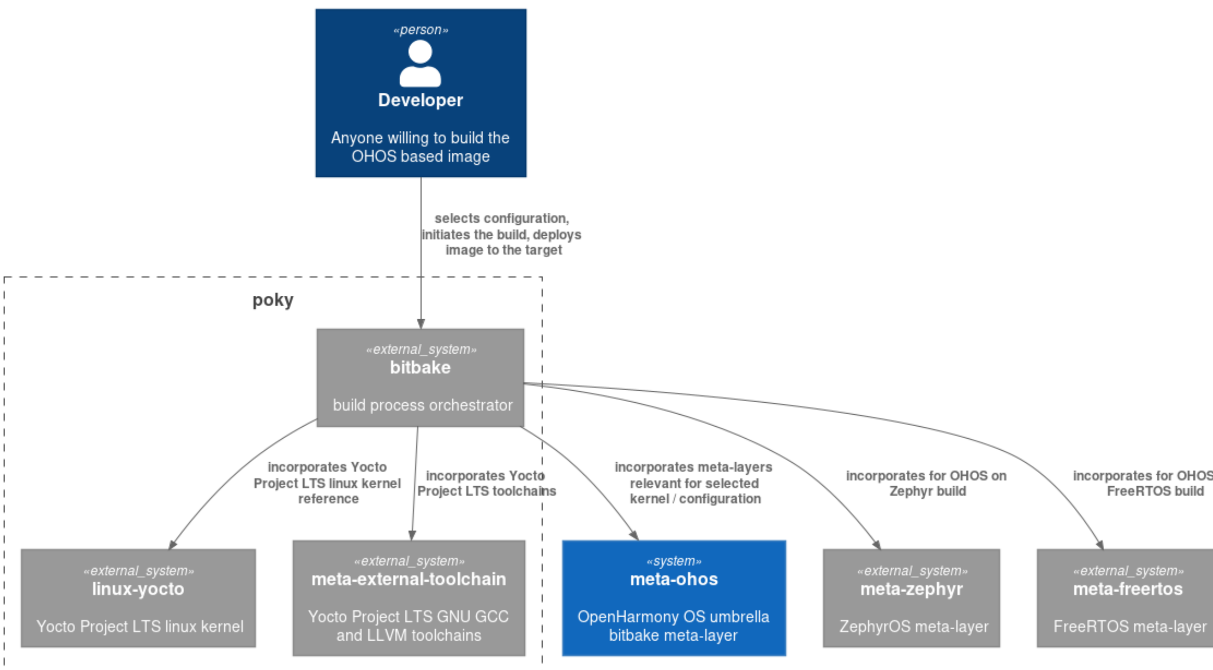
`Oniro Project` architecture is documented using c4 model.

Contents

Overview

Oniro Project build infrastructure is designed to run atop variety of OS kernels ranging from RTOSes to Linux.

`oniro` is an *umbrella* of meta layers containing build's meta-data required for compiling Oniro Project images. The architecture supports plugging various kernels.



2.3 Repo Workspace

Oniro Project uses repo to provide full workspace setup that includes all the repositories needed for building Oniro Project and developing on top.

Contents

2.3.1 The Repo Tool

In order to setup a sources workspace of Oniro Project, the `git-repo` tool is required on the host.

Oniro Project provides a patched version of the repo tool for `Ubuntu 20.04` at launchpad PPA, for `rpm/dnf` based distributions at copr, and for openSUSE, Arch and a few other distributions at OBS.

The patches are also available in the tool's source repository. One can install this tool by following the next steps:

On Ubuntu/Debian:

```
$ sudo apt-get update
$ sudo apt-get install repo
```

For Debian "contrib" repo should be [enabled](https://wiki.debian.org/SourcesList#Example_sources.list)

The repo package has not been backported to Ubuntu 20.04. PPA repository should be used for this release instead:

On Ubuntu 20.04:

```
$ sudo add-apt-repository ppa:ostc/ppa
$ sudo apt-get update
$ sudo apt-get install git-repo
```

On Distributions that Use `dnf`:

```
$ sudo dnf copr enable oniroproject/tools
$ sudo dnf --refresh install repo
```

On openSUSE or SUSE Enterprise Linux:

```
$ sudo zypper addrepo https://build.opensuse.org/project/show/home:oniroproject:tools
$ sudo zypper in repo
```

In the *zypper addrepo* line, replace `openSUSE_Tumbleweed` with the distribution you're using - a list of distributions for which the package is available here.

On Arch Linux:

Add our OBS repository to /etc/pacman.conf:

```
[openharmony-tools]
Server = http://download.opensuse.org/repositories/home:/openharmony:/tools/Arch/$arch/
```

Optionally, install the repository's signing key.

Then install the repo package with pacman.

On OpenMandriva:

OpenMandriva has already added Oniro Project version of repo to its official repositories. If you're on OpenMandriva, simply dnf install repo.

2.3.2 The Manifests

The repo manifest files are part of the main oniro repository and are to be used for configuring a workspace. The project provides a single default.xml manifest file in each of the active branches.

Depending on the specific branch of the above-mentioned repository, that manifest may either allow some projects to follow changes by selecting git branches or pin all projects to a specific git commit revision.

2.3.3 Setting-up the Workspace

Once the repo tool is installed, you can initialize and populate the workspace. This will bring in all the needed sources for building Oniro Project:

```
$ mkdir oniroproject; cd oniroproject
$ repo init -u https://booting.oniroproject.org/distro/oniro
$ repo sync --no-clone-bundle
```

2.3.4 Workspace Structure

A fully set workspace, will provide a structure similar to:

```
./oniroproject/
├── bitbake
├── docs
├── ip-policy
├── meta-openembedded
├── meta-raspberrypi
├── meta-zephyr
├── <various bitbake layers>
├── oe-core
├── oniro
└── README.md
```

All the bitbake layers are included at the root of the workspace. See for example meta-openembedded above.

It is recommended to use the root of the workspace for the build directories using build- as directory name prefix.

2.4 Oniro Project Build Flavours

Oniro Project provides default build configuration for each supported type of kernels. Each set of such configuration is called a `flavour`.

2.4.1 Overview of Build Flavours

Oniro Project can be hosted on top of variety of kernels. Currently supported kernels are Linux, Zephyr and FreeRTOS (experimental). The build system requires build configuration that is specific to each kernel and Oniro Project provides all this configuration as build templates. See Yocto documentation for more info about the underlying mechanism.

In essence, a flavour is a build configuration that was used at build initialization time for a specific kernel by passing the associated `TEMPLATECONF` configuration.

All the available `flavours` are available as subdirectories of the `flavours` directory in the root of the `oniro` repository.

Generically, when configuring a new build, one should pass the flavour as `TEMPLATECONF` to the `oe-init-build-env` script:

```
$ TEMPLATECONF=../oniro/flavours/<FLAVOUR_NAME> . ./oe-core/oe-init-build-env <BUILD_
↪NAME>
```

Notes:

- The command assumes that the working directory is the root of the repo workspace when issuing the above command.
- The variables marked in <> are to be replaced accordingly.
- Mind the `../` prefix for `TEMPLATECONF`. This is because the path provided needs to be relative to the build directory.

Once the build was initilized, you are dropped in a build environment where you have access to the `OE` tools:

```
$ bitbake <TARGET/IMAGE-NAME>
```

2.4.2 Linux Kernel Build Flavour

Oniro Project Linux build flavour is based on *oniro-linux* distribution (distro configuration).

Supported images:

- oniro-image-base
- oniro-image-base-tests
- oniro-image-extra
- oniro-image-extra-tests

Supported machines (default in **bold**):

- **qemux86-64**
- qemux86
- qemuarm
- qemuarm64

- seco-intel-b68 (SECO SBC-B68)
- stm32mp1-av96 (96Boards Avenger96)
- seco-imx8mm-c61 (SECO SBC-C61)

Build steps example:

```
$ TEMPLATECONF=../oniro/flavours/linux . ./oe-core/oe-init-build-env build-oniro-linux
$ bitbake oniro-image-base
```

You can test the image built for the qemux86-64 target by issuing:

```
$ runqemu qemux86-64 oniro-image-base wic
```

After successful bootup, you will be dropped into a login shell:

```
qemux86-64 login:
```

Default login is *root* without a password.

After login you will see the shell prompt:

```
root@qemux86-64:~#
```

To exit qemu, you can either shut down the system:

```
root@qemux86:~# poweroff -f
```

or close qemu using a key combination: *Ctrl-a followed by 'x'*.

2.4.3 Zephyr Kernel Build Flavour

Oniro Project Zephyr build flavour is based on *oniro-zephyr* distribution (distro configuration).

Supported images:

- Zephyr comes with multiple sample applications. Take a look into `sources/meta-zephyr/recipes-kernel/zephyr-kernel/` to see available recipes. You can extend them by adding recipes that use sample applications provided with Zephyr or your own applications.

Supported machines (default in **bold**):

- **qemu-x86**
- qemu-cortex-m3
- 96b-nitrogen (96Boards Nitrogen)
- 96b-avenger96 (96Boards Avenger96)
- arduino-nano-33-ble (Arduino Nano 33 BLE and Arduino Nano 33 BLE Sense)
- nrf52840dk-nrf52840 (Nordic Semiconductor nRF 52840 Development Kit)

Build steps example:

```
$ TEMPLATECONF=../oniro/flavours/zephyr . ./oe-core/oe-init-build-env build-oniro-zephyr
$ bitbake zephyr-philosophers
```

You can test the image built for the qemu-x86 target by issuing:

```
$ runqemu qemu-x86
```

After successful bootup, the output of the application will be similar to:

```
Booting from ROM..*** Booting Zephyr OS build zephyr-v2.4.0  ***
Philosopher 0 [P: 3]  THINKING [  300 ms ]
Philosopher 1 [P: 2]   EATING  [  575 ms ]
Philosopher 2 [P: 1]        STARVING
Philosopher 3 [P: 0]   EATING  [  525 ms ]
Philosopher 4 [C: -1]  THINKING [  475 ms ]
```

To exit qemu, use the following key combination: *Ctrl-a followed by 'x'*.

2.4.4 FreeRTOS Kernel Build Flavour

Oniro Project FreeRTOS build flavour is based on *freertos* distribution (distro configuration).

Supported images:

- freertos-demo

Supported machines (default in **bold**):

- **qemuarmv5**

Build steps example:

```
$ TEMPLATECONF=../oniro/flavours/freertos . ./oe-core/oe-init-build-env build-oniro-
↪freertos
$ bitbake freertos-demo
```

You can test the image built for the qemuarmv5 target by issuing:

```
$ runqemu qemuarmv5
```

After successful bootup, the output of the application will be similar to:

```
###### - FreeRTOS sample application - ######

A text may be entered using a keyboard.
It will be displayed when 'Enter' is pressed.

Periodic task 10 secs
Waiting For Notification - Blocked...
Task1
Task1
You entered: "HelloFreeRTOS"
Unblocked
Notification Received
Waiting For Notification - Blocked...
```

To exit qemu, use the following key combination: *Ctrl-a followed by 'x'*.

2.5 Build Configuration

The build system's recipes provide various functionalities that expose knobs and primitives.

2.5.1 Build System Visual Customizations

Contents

Weston Dynamic Configuration

The build exposes mechanism to tweak weston configuration through build variables. These variables can be provided as part of any configuration (eg. *local.conf*, *distro.conf*).

The mechanism is enabled by setting `WESTON_DYNAMIC_INI` to `1`. Any of the following variables will be ignored if this variable is not set to `1`. The configuration file path can also be set via a variable: `WESTON_INI_PATH`. The default value of `WESTON_INI_PATH` should be fine for most of the cases.

Additional variable to be used in conjunction with `WESTON_DYNAMIC_INI`:

- `WESTON_INI_NO_TOOLBAR` - remove the shell panel when set to `1`
- Configuration for shell background * `WESTON_INI_BACKGROUND_IMAGE` - sets shell.background-image accordingly * `WESTON_INI_BACKGROUND_COLOR` - sets shell.backgound-color accordingly * `WESTON_INI_BACKGROUND_TYPE` - sets shell.background-type accordingly

Epiphany Support for Application Mode

Epiphany is one of the browsers supported by the build meta-data. It provides a webkitgtk-based browser.

The build exposes the ability to run the browser as a system service in application mode. This can be easily configurable and extended via the build metadata and variables.

Available variables:

- `EPIPHANY_APP` - the application name
- `EPIPHANY_URL` - the URL to be used when browser starts
- `EPIPHANY_RDEPENDS` - additional dependencies needed at runtime
- `EPIPHANY_SERVICE_ENABLED` - when set to `1`, build system will enable the systemd service for starting at boot

The build system provides support for using this mechanism with *HomeAssistant*. See this support as an example for how to implement a custom application mode for Epiphany.

2.6 Operating System

Oniro Project provides support for a set of kernels. This documentation details various aspects for each kernel type.

2.6.1 Oniro Project - Linux

Disk Partition Table

Contents

Overview

The OS defines the partitions included as part of the Linux-based distro as it follows:

- boot
  - filesystem label: x-boot (partition name when relevant)
  - It provides boot artefacts required by the lower bootloader assumptions. It is device-specific both in terms of filesystem and content.
- sys-a
  - filesystem label: x-sys-a (partition name when relevant)
  - It provides the root filesystem hierarchy.
  - Filesystem type, configuration and structure are device-independent.
  - This partition is the only one provided with a redundant counterpart (see below).
- sys-b
  - filesystem label: x-sys-b (partition name when relevant)
  - It provides a redundant root filesystem hierarchy used as part of the system update strategies.
  - Filesystem type, configuration and structure are device-independent.
- dev-data
  - filesystem label: x-dev-data (partition name when relevant)
  - Device-specific data meant to be preserved over system reset (factory reset).
  - The runtime will completely treat this data read-only.
- sys-data
  - filesystem label: x-sys-data (partition name when relevant)
  - This partition holds the system state to deal with the root filesystem as a read-only asset.
  - It ties closely into the system update strategies.

  - Data is kept over system updates (subject to state transition hooks) but discarded over factory reset.

- app-data
  - filesystem label: x-app-data (partition name when relevant)
  - This partition provides application data storage.
  - Data is kept over system updates (subject to state transition hooks) but discarded over factory reset.

The build system tries to unify the partition as much as possible, leaving upper layers (for example, the system update layers) with as few deviations to deal with as feasible. This means that filesystem labels and partition names are to be assumed by the OS components.

Partition Table

The OS will support both MBR and GPT as partition table type. In this way, the OS can achieve more extensive device support.

The OS assumes a GPT disk layout as it follows:

- 4MiB is left untouched at the start of the disk (to accommodate for hardware-specific requirements).
- All partitions are aligned to 4MiB.
- The filesystem labels and partition names are as described above.

On the MBR side, the disk layout is similar to GPT. The design mainly workarounds the four physical partitions limitation:

- 4MiB is left untouched at the start of the disk (to accommodate for hardware-specific requirements).
- All partitions are aligned to 4MiB.
- The filesystem labels and partition names are as described above.
- The 4th partition is defined as extended and contains all the data partitions (dev-data, sys-data and app-data).

2.7 Continuous Integration

2.7.1 Machines and Flavours

The following GitLab job definitions are included by the central shared pipelines in the manifest repository and constitute the set of supported combination of `FLAVOUR` and `MACHINE`.

> **Warning:** Do not include YAML files from the oniro repository directly. The primary entry point for build and test pipelines is defined by the manifest repository.

linux-qemu-x86

This job extends *.build-linux* job from the manifest repository and builds `oniro-image-base-tests` and `oniro-image-extra-tests` using the Linux flavour of Oniro Project and `MACHINE=qemux86`. This job checks that Oniro Project software can be built for a basic 32bit x86 virtual machine.

The cache for this job is publicly available.

linux-qemu-x86_64

This job extends *.build-linux* job from the manifest repository and builds `oniro-image-base-tests` and `oniro-image-extra-tests` using the Linux flavour of Oniro Project and `MACHINE=qemux86-64`. This job checks that Oniro Project software can be built for a basic 64bit x86 virtual machine.

The cache for this job is publicly available.

linux-seco-intel-b68

This job extends *.build-linux* job from the manifest repository and builds `oniro-image-base-tests` and `oniro-image-extra-tests` using the Linux flavour of Oniro Project and `MACHINE=seco-intel-b68`. This job checks that Oniro Project software can be built for the SECO B68 development board, which contains an Intel x86_64 SoC.

Note: The cache for this job is not public, pending legal review of any firmware that may be included.

linux-seco-imx8mm-c61

This job extends *.build-linux* job from the manifest repository and builds `oniro-image-base-tests` and `oniro-image-extra-tests` using the Linux flavour of Oniro Project and `MACHINE=seco-imx8mm-c61`. This job checks that Oniro Project software can be built for the SECO C61 development board, which contains the NXP i.MX 8M Mini SoC, which implements 64bit ARMv8 architecture.

Note: The cache for this job is not public, as it contains proprietary elements that cannot be redistributed without an agreement with Freescale.

linux-stm32mp1-av96

This job extends *.build-linux* job from the manifest repository and builds `oniro-image-base-tests` and `oniro-image-extra-tests` using the Linux flavour of Oniro Project and `MACHINE=stm32mp1-av96`. This job checks that Oniro Project software can be built for the 96Boards Avenger development board, which contains the STM32MP157 SoC, which implements 32bit ARMv7 architecture.

Note: The cache for this job is not public, pending legal review of any firmware that may be included.

linux-raspberrypi4-64

This job extends *.build-linux* job from the manifest repository and builds `oniro-image-base-tests` and `oniro-image-extra-tests` using the Linux flavour of Oniro Project and `MACHINE=raspberrypi4-64`. This job checks that Oniro Project software can be built for the Raspberry Pi 4B development board, which contains the BCM2711 SoC, which implements 64bit ARMv8 architecture.

Note: The cache for this job is not public, pending legal review of any firmware that may be included.

zephyr-qemu-x86

This job extends *.build-zephyr* job from the manifest repository and builds `zephyr-philosophers` using the Zephyr flavour of Oniro Project and `MACHINE=qemu-x86`. This job checks that Zephyr can be built for a basic 32bit x86 virtual machine.

The cache for this job is publicly available.

zephyr-qemu-cortex-m3

This job extends *.build-zephyr* job from the manifest repository and builds `zephyr-philosophers` using the Zephyr flavour of Oniro Project and `MACHINE=qemu-cortex-m3`. This job checks that Zephyr can be built for a basic 32bit ARM micro-controller virtual machine.

The cache for this job is publicly available.

zephyr-96b-nitrogen

This job extends *.build-zephyr* job from the manifest repository and builds `zephyr-philosophers` using the Zephyr flavour of Oniro Project and `MACHINE=96b-nitrogen`. This job checks that Zephyr can be built for the 96Boards Nitrogen development board, which contains an nRF52832 SoC.

Note: The cache for this job is not public, pending legal review of any firmware that may be included.

zephyr-96b-avenger

This job extends *.build-zephyr* job from the manifest repository and builds `zephyr-philosophers` using the Zephyr flavour of Oniro Project and `MACHINE=96b-avenger96`. This job checks that Zephyr can be built for the 96Boards Avenger development board cortex-M3 core, embedded into STM32MP157 SoC.

Note: The cache for this job is not public, pending legal review of any firmware that may be included.

zephyr-arduino-nano-33-ble

This job extends *.build-zephyr* job from the manifest repository and builds `zephyr-philosophers` using the Zephyr flavour of Oniro Project and `MACHINE=arduino-nano-33-ble`. This job checks that Zephyr can be built for the Arduino Nano 33 BLE development board Cortex-M4 core, embedded into nRF 52840 SoC.

Note: The cache for this job is not public, pending legal review of any firmware that may be included.

freertos-armv5

This job extends *.build-freertos* job from the manifest repository and builds `freertos-demo` using the FreeRTOS flavour of Oniro Project and `MACHINE=qemuarmv5`. This job checks that FreeRTOS can be built for a basic 32bit ARMv5 micro-controller virtual machine.

The cache for this job is publicly available.

2.7.2 Special Jobs

linux-glibc-qemu-x86_64

This job extends `linux-qemu-x86_64` and differs in the following way.

This job performs a build with the libc switched to `glibc`. It only runs on a schedule that is defined in the GitLab project settings for the *manifest* repository. In practice it runs daily to check if the Linux favour could be switched back to `glibc`, from the default `musl` that is used right now.

2.7.3 Hidden Jobs

There's a number of *hidden jobs*, which start with the dot character, that are used as foundation for the set of *Machines and Flavours*. Hidden jobs do not participate in any pipeline directly. They can only be used as templates, using the `extends: ...` mechanism, to share and reuse implementation details.

.workspace

The `.workspace` job assembles a *git-repo* workspace, as described by *manifest* file. This job is a foundation for other jobs.

This workspace is **not** constructed in `$CI_PROJECT_DIR`, which is by GitLab runner. To avoid clashes with any files that may be present there, it is constructed in a temporary directory. Additional logic stores and restores the path of that directory between the code in `before_script`, `script` and `after_script` as those execute in separate shell processes.

Constraints

This job uses the `ostc-builder` container image, mainly for convenience, as it is often extended to perform other tasks, such as bitbake builds. Other containers can be used, as long as they have the `repo` program pre-installed.

In addition this job uses the `large-disk` tag to be scheduled on a machine with access to a shared NFS volume with git mirrors, that greatly speed up the process of constructing the workspace from scratch.

Variables

CI_ONIRO_INSTANCE_SIZE

An arbitrary GitLab Runner tag selecting the size of a system instance which processes the .workspace job, or its derivative. This defaults to s3.large.8 which translates to two cores and 16GB of memory.

This can be used to route specific jobs to specific instance sizes using easy-to-use variable map, rather than more painful to use tag list.

CI_ONIRO_RUNNER_TAG

An arbitrary GitLab Runner tag selecting the runner which processes the .workspace job, or its derivative. This defaults to an empty string but can be used to pick a specific GitLab runner, as long as that runner has the matching tag set.

This can be used to perform scheduled builds in a non-default location, as long as the pipeline schedule defines this variable.

CI_ONIRO_MANIFEST_URL

The URL consumed by git-repo. You only want to change this if you forked the entire infrastructure and want to use it in private.

The default value is `https://booting.oniroproject.org/distro/oniro`.

If you change the default value, please set `CI_ONIRO_MANIFEST_MIRROR` as well.

CI_ONIRO_MANIFEST_BRANCH

The name of the git branch of the manifest repository. Unless special circumstances apply this does not need to be customized.

For testing changes coming into the *manifest* repository itself, use `$CI_COMMIT_REF_NAME`, which will check out the manifest as described by the specific commit being tested.

The default value is `develop`.

CI_ONIRO_MANIFEST_NAME

The name of the manifest file from the repository mentioned above.

The default value is `default.xml`.

CI_ONIRO_MANIFEST_MIRROR

Name of the `git-repo` mirror matching to use, expressed as a directory name in the shared disk volume. Check the section about git-repo mirror below, for details.

The default value is `ostc-develop`.

CI_ONIRO_GIT_REPO_PATH

The path of the git repository to deviate from what the git-repo manifest describes. This variable should be used for constructing CI jobs for repositories directly described by the manifest.

When set to a non-empty value, the specified git repository will be first checked out by the git-repo according to what is described in the manifest, and then changed again, to point to `$CI_COMMIT_SHA`. The logic in the script supports the forked repository workflow, by reusing the merge setup GitLab does on pipelines for merge results. See GitLab documentation on Pipelines for Merge Results for more information.

The default value is the empty string.

Local git-repo Mirror

The `.workspace` job relies on a git-repo mirror that is mounted into the execution environment provided by the GitLab worker. The mirror is created and kept up-to-date by the scheduled run of the pipeline in ostc-manifest-mirror repository

When the mirror is out-of-date additional git revisions needed to construct the workspace are fetched form the respective upstream repositories. Some of the git repositories described by the manifest are rather large and are hosted on infrastructure outside of the OSTC cloud provider, making this an important optimization.

The mirror is automatically published in https://cache.ostc-eu.org/git-repo-mirrors/, with two specific mirrors being available: *ostc-dev* and *gitee-dev*.

.bitbake-workspace

The `.bitbake-workspace` job extends the *.workspace* job to configure and initialize BitBake for the desired build operations. The job does not build anything by itself, that functionality is available in the `.build-recipe` job.

Job Variables

The `.bitbake-workspace` job defines several variables as a way to customize the way BitBake is configured.

CI_ONIRO_BUILD_FLAVOUR

The name of the *flavour* of Oniro OS, which effectively picks the kernel type. This is used to select the initial BitBake configuration template. Templates are stored in the **oniro** repository.

Available values are `linux`, `zephyr` and `freertos`. There is no default value. This variable must be set by a derivative job, it is usually set by the three `.build-linux`, `.build-zephyr` and `.build-freertos` jobs. Specific build jobs, in turn, extend those.

CI_ONIRO_BUILD_CACHE

The set of build caches to use.

Currently there are only two sets `private` or `pub`. The names directly correspond to file system path where the cache is stored.

By default all builds are assumed to be tainted, and use the private cache. If a given build configuration is not legally problematic, in the sense of pulling in code or blobs that are non-redistributable, this attribute can be set to `pub`.

Public build cache is exposed as https://cache.ostc-eu.org/ and can be used by third parties to speed up local builds.

The default value is `private`.

CI_ONIRO_DEVTOOL_RECIPE_NAME

Name of the BitBake recipe to upgrade.

This may be set by jobs which extend or re-define the `.bitbake-workspace` job. It must be used in tandem with `CI_ONIRO_DEVTOOL_LAYER_PATH`. The side effects are discussed below.

CI_ONIRO_DEVTOOL_LAYER_PATH

Path of the layer containing the recipe to upgrade.

This may be set by jobs which extend or re-define the `.bitbake-workspace` job. It must be used in tandem with `CI_ONIRO_DEVTOOL_RECIPE_NAME`.

Before the build is attempted, the recipe is updated to ignore any existing patches and point to the code corresponding to the repository and commit being tested.

```
devtool upgrade \
    --no-patch \
    --srcrev "$CI_COMMIT_SHA" \
    --srcbranch "$CI_COMMIT_REF_NAME" \
    "$CI_ONIRO_DEVTOOL_RECIPE_NAME"
devtool finish \
    --remove-work \
    --force \
    "$CI_ONIRO_DEVTOOL_RECIPE_NAME" \
    "$(basename "$CI_ONIRO_DEVTOOL_LAYER_PATH")"
```

This functionality is useful for testing incoming changes to repositories that contain source code that is already packaged one of the layers.

Configuring BitBake

The `local.conf` file can define numerous variables that influence the BitBake build process. This job offers a declarative method of doing that. Job variables with have the prefix `CI_ONIRO_BB_LOCAL_CONF_` are converted to `attr = "value"` and those with prefix `CI_ONIRO_BB_LOCAL_CONF_plus_equals_` are converted to `attr += "value"`.

This method is friendly to job inheritance and re-definition. Derivative jobs can add or re-define variables without having to duplicate any imperative logic or maintaining synchronized settings across distinct jobs.

This mechanism is used to set the following BitBake variables

CONNECTIVITY_CHECK_URIS

BitBake contains a connectivity check system. That system relies on access to `https://example.com/`. OSTC cloud provider has a DNS configuration problem, where that specific domain is not resolved correctly. For a compatible workaround the connectivity check URL is set to `https://example.net/`.

This is implemented using the `CI_ONIRO_BB_LOCAL_CONF_` system.

DL_DIR

BitBake downloads source archives and git repositories in the through the `fetch` task of many recipes. Those are all stored in the local file system and can be shared between separate builds for a great speed up, as the files are obtained from multiple third party servers and their connectivity varies.

To optimize the build process, the download directory is set to point at the shared NFS volume that persists between job execution, and is more efficient than the artifact system, that copies all the data regardless of the need to actually use that data in practice.

The default location is changed to `/var/shared/$CI_ONIRO_BUILD_CACHE/bitbake/downloads`, but this should be treated as an implementation detail. The location may change in the future. The download cache is not automatically purged yet. In the future it may be purged periodically, if space becomes an issue.

Note that the location relies on the value of `$CI_ONIRO_BUILD_CACHE` discussed above.

SSTATE_CACHE

BitBake relies on an elaborate cache system, that can be used to avoid duplicating work at the level of a specific recipe. The dependencies and side-effects of each recipe are recorded in the cache, and are reused whenever possible.

Having access to a persistent cache has a dramatic effect on the performance of the CI system as, in the fast-path, it can avoid virtually all compilation tasks and simply assemble the desired system image out of intermediate files present in the cache.

The default location is changed to `/var/shared/$CI_ONIRO_BUILD_CACHE/bitbake/sstate-cache`, but this should be treated as an implementation detail. The location may change in the future. The sstate cache is not automatically purged yet. It can be purged periodically with the only caveat, that initial builds will be much slower.

Cache Considerations

The `.bitbake-workspace` job configures BitBake to use a persistent directory that is shared between CI jobs, for the location of the `download` directory as well as the `sstate-cache` directory.

The job is using GitLab runner tags to schedule jobs in the environment where that shared storage is available. When a new dependency is added or when the layers and recipes are changed or updated, the download is automatically populated with the necessary source archives. Similarly `sstate-cache` is populated by all the build jobs present throughout the CI system.

Due to legal restrictions, the caches are split into two pairs, public and private. The public cache is automatically published in https://cache.ostc-eu.org/bitbake/ The private cache, which is used by default, is available on the same volume but it is not shared anywhere.

In case the cache is fed with a software package that is, in retrospective somehow problematic, for example, by not being freely redistributable, the cache can be purged at will.

For details on how cache selection and BitBake configuration looks like, please refer to the pipeline source code.

.build-linux

The `.build-linux` job extends the *.bitbake-workspace* job. It sets `CI_ONIRO_BUILD_FLAVOUR` to `linux` and builds the bitbake targets (e.g. images) as defined by `CI_ONIRO_BITBAKE_TARGETS` (defaults included).

The images are built one after another, allowing CI to fail quickly in case of any problems with the more fundamental image. The set of built images may change over time, as additional reference images are defined.

Usage Guide

This job is not intended for direct use. Instead it serves as a base for all the Linux-specific *Machines and Flavours*. It may be re-defined in a pipeline to alter `rules` or `variables` in a way that fits a particular purpose.

If used directly, it is recommended pick the desired `MACHINE` and to override the entire script section and refer to the base `.bitbake-workspace` job as illustrated below. The set of build operations can then be tailored to the purpose of the desired job.

```yaml
build-something-linux-specific:
  extends: .build-linux
  variables:
    MACHINE: "..."
  script:
    - !reference [.bitbake-workspace, script]
    - true # put your code here
```

Note: `CI_ONIRO_BITBAKE_TARGETS` can be overwritten if defaults are not desired.

.build-linux-matrix

The `.build-linux-matrix` job extends the *.build-linux* job and otherwise behaves exactly the same but builds each of the `CI_ONIRO_BITBAKE_TARGET` as a separate job. This creates a wider pipeline which unlocks additional parallelism.

Usage Guide

Note that due to the way parallel matrix jobs are defined, to change the set of bitbake recipes to build you must re-define the `parallel/matrix` element entirely. Changing `CI_ONIRO_BITBAKE_TARGETS` is not effective.

.build-zephyr

The `.build-zephyr` job extends the *.bitbake-workspace* job. It sets `CI_ONIRO_BUILD_FLAVOUR` to `zephyr` and builds the bitbake targets (e.g. images) as defined by `CI_ONIRO_BITBAKE_TARGETS` (defaults included).

Usage Guide

This job is not intended for direct use. Instead it serves as a base for all the Zephyr-specific *Machines and Flavours*.

.build-freertos

The `.build-freertos` job extends the *.bitbake-workspace* job. It sets `CI_ONIRO_BUILD_FLAVOUR` to `freertos` and builds the `freertos-demo` test image.

Usage Guide

This job is not intended for direct use. Instead it serves as a base for all the FreeRTOS-specific *Machines and Flavours*.

.build-recipe

The `.build-recipe` job extends the *.bitbake-workspace* job to build a single BitBake recipe. The recipe is built and all the results are discarded.

Variables

CI_ONIRO_RECIPE_NAME

The name of the recipe to build.

There is no default value, this must be set by the extending job.

Usage Guide

To use this job in your pipeline include the generic build definition file, `build-generic.yaml` and define a job with the minimal configuration, as illustrated below:

```
build-something:
  extends: .build-recipe
  variables:
    CI_ONIRO_BUILD_FLAVOUR: "..."
    CI_ONIRO_RECIPE_NAME: "..."
    MACHINE: "..."
```

Pick the desired `CI_ONIRO_BUILD_FLAVOUR`, `CI_ONIRO_RECIPE_NAME` and `MACHINE`. For discussion of `CI_ONIRO_BUILD_FLAVOUR` refer to the *.bitbake-workspace* job. `CI_ONIRO_RECIPE_NAME` is any BitBake recipe available in the selected flavour. `MACHINE` is desired machine name. Supported values are documented alongside each flavour in the *oniro* repository.

.build-image

The `.build-image` job extends the *.build-recipe* job to build a single recipe, which should describe an image, and to collect the resulting image and licenses as job artifacts.

Usage Guide

This job is configured exactly the same as *.build-recipe*.

Implementation Details

The job handles differences between the name of the temporary build directory between various Oniro OS flavours. Internally BitBake is interrogated for the value of `TMPDIR` and the image is copied back to a subdirectory of `$CI_PROJECT_DIR` for delivery to the GitLab runner.

.build-wic-image

The `.build-wic-image` job extends the *.build-image* job to collect only the **.wic.** and **.bmap* files and remove all the other files that would normally be collected by the artifact system. It is recommended for Linux builds which produce wic images, as the size of subset of collected artifacts is considerably smaller than what *.build-image* provides.

Usage Guide

This job is configured exactly the same as *.build-image* and *.build-recipe*.

.build-docs

The `.build-docs` job builds reStructuredText documentation.

This job offers no configuration and works by convention. This is done to both simplify its use and improve uniformity of developer experience. The job automatically reacts to changes in the `docs/` directory as well as the definition of the pipeline. Built documentation is collected as job artifacts.

Usage Guide

To use this job in your pipeline include the generic build definition file, `build-generic.yaml` and define a `build-docs` job:

```
build-docs:
  extends: .build-docs
```

The job expects the project to have the following documentation structure.

```
└── docs
    ├── index.rst
    └── Makefile
```

The `Makefile` should build the documentation into the `build` directory, relative to the `docs` directory. A sample, re-usable example is is provided below:

```
# SPDX-FileCopyrightText: Huawei Inc.
#
# SPDX-License-Identifier: Apache-2.0

.PHONY: all
all:
	sphinx-build -W -C \
		-D html_theme=sphinx_rtd_theme \
		-D project='Name of the project' \
		-D copyright='Copyright Holder' \
		. build

clean:
	rm -rf ./build
```

.lava-test

The `.lava-test` job creates the LAVA test job definition based on the pipeline information and submits the job to LAVA.

The artifact of this job is the list of LAVA job id(s) submitted by this job. This will be consumed by the *.lava-report*

Job Variables

The `.lava-test` job defines several variables as a way to customize the LAVA job definition before submiting the job to LAVA.

CI_LAVA_JOB_DEFINITION

This is the url to the job definition template. The template needs to have variables inside which will be replaced by this job. The list of variables is the following:

- $ci_pipeline_id - pipeline ID of the CI job ($CI_PIPELINE_ID in GitLab)
- $ci_job_id - CI job ID ($CI_JOB_ID in GitLab)
- $ci_project_id - CI project ID ($CI_PROJECT_ID in GitLab)
- $build_job_id - job ID of the **build** job within the same pipeline
- $callback_url - url which triggers the execution or the manual job which collects the results back from LAVA to GitLab. See *.lava-report*

CI_BUILD_JOB_NAME

The name of the job that builds the artifacts which will be used by LAVA to boot up the DUT for this particular LAVA job. If we test multiple builds in one CI job, each will have different build job name.

CI_REPORT_JOB_NAME

The name of the `report` job which will be triggered manually when the LAVA job(s) are finished with executution. This job will collect the results from LAVA and import them to GitLab.

CI_LAVA_INSTANCE

The base url of the LAVA server. The variable should be added to GitLab CI/CD variables.

CI_LAVA_TOKEN

The authentication token generated on LAVA server for job submission from gitlab. The variable should be added to GitLab CI/CD variables.

.lava-report

The `.lava-report` job iterates through the submitted jobs from *.lava-test* and collects the artifacts from these jobs via LAVA REST API.

Job Variables

CI_LAVA_INSTANCE

The base url of the LAVA server. The variable should be added to GitLab CI/CD variables.

.aggregate-docs

The `.aggregate-docs` job triggers a build of the aggregated documentation of Oniro OS.

Oniro OS documentation is maintained in an unusual way. Individual repositories contain dedicated documentation that can be built with :doc`build-docs` job. A special *centralized* documentation project aggregates documentation from multiple git repositories, as checked out by *git-repo* based on the *manifest* file, to build a standalone document.

This job encapsulates knowledge on how to trigger the aggregated build upon changes to documentation specific to a given project.

Usage Guide

> **Warning:** If a project is pinned in the *git-repo manifest* then using this job makes no sense, as the resulting aggregate will not change without a prior modification of the manifest.

To use this job in your pipeline include the generic build definition file, `build-generic.yaml` and define the `aggregate-docs` job in addition to the *.build-docs* job.

```yaml
# Build documentation present in the repository.
build-docs:
  extends: .build-docs

# Trigger aggregation of Oniro OS documentation.
aggregate-docs:
  extends: .aggregate-docs
  needs: [build-docs]
```

2.8 Supported images

To create a custom Linux distribution to match the product requirements, Oniro Project includes a set of predefined images for developing a product image.

2.8.1 Linux Kernel

The Linux kernel is a free and open-source Unix-like operating system (OS) kernel that serves as the primary interface between the computer's hardware and its processes.

Oniro Project supports the following images listed in the table:

Table 1: Linux supported images

Image Name	Description
oniro-image-base	• Oniro Project image including the base OS software stack. • This image also includes middleware and application packages to support a wide range of hardware which includes WiFi, Bluetooth, sound, and serial ports.
oniro-image-extra	• Oniro Project Wayland image including the base OS software stack. This is a Wayland protocol and Weston reference compositor-based image. • It uses the Wayland protocol and implementation to exchange data with its clients. • This image provides the Wayland protocol libraries and the reference Weston compositor and includes a Wayland-capable terminal program.

To build a Linux-based image for a supported machine, see *Linux Kernel Build Flavour*.

2.8.2 Zephyr Kernel

The Zephyr OS is a well-known security-oriented real-time operating system (RTOS) that is intended for use on resource-constrained and embedded systems.

For more detailed information on Zephyr OS Kernel, see Zephyr documentation.

Oniro Project supports the following images for the Zephyr OS kernel:

Table 2: Zephyr supported images

Image Name	Description
zephyr-philosophers	A sample Zephyr application implementing the Dining Philosophers problem.

To build a Zephyr-based image for a supported machine, see *Zephyr Kernel Build Flavour*.

2.8.3 FreeRTOS Kernel

The FreeRTOS kernel is a real-time operating system (RTOS) that runs on a variety of platforms which is used to build microcontroller-based embedded applications.

The standard RTOS kernel binary image ranges from 4000 to 9000 bytes. Oniro Project supports the following images for FreeRTOS Kernel:

Table 3: FreeRTOS supported images

Image Name	Description
freertos-demo	Machine configuration for running an ARMv5 system on QEMU.

To build a FreeRTOS-based image for a supported machine, see *FreeRTOS Kernel Build Flavour*.

2.9 Hardware Support in Oniro Project

This section details the hardware (including virtualized) supported as part of Oniro Project.

2.9.1 Supported Boards

This section details the boards supported as part of Oniro Project.

96Boards Avenger96

Contents

Overview

Avenger96 is a STM32MP157xx (Cortex-A7 + Cortex-M4) development board designed by the 96Boards initiative. Due to presence of the application processors and the microcontroller, Avenger96 can simultaneously run Linux and Zephyr kernels. The application processor is responsible for powering up and programming the microcontroller with the appropriate image. Linux provides interfaces to communicate with the program running on the microcontroller.

Hardware

- For detailed specification, see Avenger96 product page on the 96Boards website.
- For hardware user manual and schematics, see 96Boards GitHub documentation repository.

For more details on Avenger96 board, see Avenger96 product page.

Working with the Board

Building an Oniro image

To clone the source code, perform the procedure in: *Setting up a repo workspace*.

Linux image

1. Source the environment with proper template settings, flavour being *linux* and target machine being *stm32mp1-av96*. Pay attention to how relative paths are constructed. The value of *TEMPLATECONF* is relative to the location of the build directory *./build-linux*, that is going to be created after this step:

```
$ TEMPLATECONF=../oniro/flavours/linux . ./oe-core/oe-init-build-env build-oniro-linux
```

2. You will find yourself in the newly created build directory. Call *bitbake* to build the image. For example, if you are using *oniro-image-base* run the following command:

```
$ MACHINE=stm32mp1-av96 bitbake oniro-image-base
```

To generate images for eMMC on SD card, refer to the *Flashing an Oniro image*.

Zephyr image

1. Source the environment with proper template settings, flavour being *zephyr* and target machine being *96b-avenger96*:

```
$ TEMPLATECONF=../oniro/flavours/zephyr . ./oe-core/oe-init-build-env build-oniro-zephyr
```

2. You will find yourself in the newly created build directory. Call *bitbake* to build the image. The image name is the name of the Zephyr application.

```
$ MACHINE=96b-avenger96 bitbake zephyr-philosophers
```

3. The output file will be located in the build directory *./tmp-newlib/deploy/images/96b-avenger96/*.

Flashing an Oniro image

For Linux, *bmaptool <https://github.com/intel/bmap-tools>* is recommended to create an SD card image. The images we provide also create wic files (disk images) that you can use directly. You can also use the STM32 Cube Programmer.

For Zephyr, there is no automation as for now. To have the ELF file in the filesystem:

- Copy the image manually to the filesystem using a method of your choice
- Include it in the image before flashing the card/eMMC
- Copy the file manually to the card or just *scp* it to the board after you set up networking.

Linux image

SD Card

The Avenger96 board supports multiple boot options which are selected by the DIP-switch S3. Make sure the boot switch is set to boot from the SD-Card.

To set the boot option from the SD card using DIP-switch S3, set the BOOT 0 (Switch 1) and BOOT 2 (Switch 3) to 1 and set BOOT 1 (Switch 2) to 0 on the circuit board.

For more information on Avenger96 boot options, see Getting Started with the Avenger96.

1. After the image is built, you are ready to burn the generated image onto the SD card. We recommend using *bmaptool <https://github.com/intel/bmap-tools>* and the instructions below will use it. For example, if you are building oniro-image-base run the following command replacing (or defining) `$DEVNODE` accordingly:

```
$ cd tmp/deploy/images/stm32mp1-av96
$ bmaptool copy oniro-image-base-stm32mp1-av96.wic.bz2 $DEVNODE
```

2. Put the card to the board and turn it on.

STM32 Cube Programmer

After you build the image, follow the instructions in Avenger96 Image Programming, pointing the program to the *./tmp/deploy/images/stm32mp1-av96/flashlayout_oniro-image-base/trusted/FlashLayout_emmc_stm32mp157a-av96-trusted.tsv* flash layout file.

Zephyr image

Prerequisites

- Linux is running on the board.
- Make sure that Linux is built with *remoteproc* support. To check status of remoteproc do:

```
root@stm32mp1-av96:~# dmesg | grep remoteproc
[    2.336231] remoteproc remoteproc0: m4 is available
```

1. Copy the Zephyr image to the board using a method of your choice.
2. Check what the `remoteproc` framework knows about the name and location of the firmware file. The default values are presented as follows. Empty path defaults to `/lib/firmware`:

```
root@stm32mp1-av96:~# cat /sys/module/firmware_class/parameters/path
<empty>

root@stm32mp1-av96:~# cat /sys/class/remoteproc/remoteproc0/firmware
rproc-m4-fw
```

3. Configure the name and the location to suit your needs. For example, the firmware is located in `/root/zephyr.elf`:

```
root@stm32mp1-av96:~# echo "/root" > /sys/module/firmware_class/parameters/path
root@stm32mp1-av96:~# echo "zephyr.elf" >  /sys/class/remoteproc/remoteproc0/firmware
```

4. Power up the Cortex-M4 core:

```
root@stm32mp1-av96:~# echo start > /sys/class/remoteproc/remoteproc0/state
remoteproc remoteproc0: powering up m4
remoteprocroc remoteproc0: Booting fw image rproc-m4-fw, size 591544
rproc-srm-core m4@0:m4_system_resources: bound m4@0:m4_system_resources:m4_led (ops␣
↪0xc0be1210)
remoteproc remoteproc0: remote processor m4 is now
```

5. Firmware output can be inspected with:

```
root@stm32mp1-av96:~# cat /sys/kernel/debug/remoteproc/remoteproc0/trace0
Philosopher 5 [C:-2]        STARVING
Philosopher 3 [P: 0]    DROPPED ONE FORK
Philosopher 3 [P: 0]  THINKING [  25 ms ]
Philosopher 2 [P: 1]   EATING  [  425 ms ]
Philosopher 3 [P: 0]        STARVING
Philosopher 4 [C:-1]        STARVING
Philosopher 4 [C:-1]    HOLDING ONE FORK
Philosopher 4 [C:-1]   EATING  [  800 ms ]
Philosopher 3 [P: 0]    HOLDING ONE FORK
Philosopher 2 [P: 1]    DROPPED ONE FORK
Philosopher 2 [P: 1]  THINKING [  725 ms ]
Philosopher 1 [P: 2]   EATING  [  225 ms ]
```

There is no fully-featured console available in Linux yet, so typing commands to the Zephyr application is not possible.

Testing the Board

Serial Port

To connect the USB converter serial port to the low-speed connector, see Hardware User Manual.

Warning:

- The low speed connector is 1.8V tolerant, therefore the converter must be 1.8V tolerant.
- Do not connect 5V or 3.3V tolerant devices to the connector to avoid SoC damage.

Ethernet

Wired connection works out of the box. You can use standard tools like ip, ifconfig to configure the connection. The connection seems to have stable 1Gb/s bandwidth.

For any fault in the hardware device, see *How to handle faulty hardware device*.

USB Host

Just plug something to the USB port. The board seems to work fine with an external 500GB USB 3.0 HDD.

```
root@stm32mp1-av96:~# lsusb
Bus 002 Device 003: ID 0930:0b1f Toshiba Corp.
Bus 002 Device 002: ID 0424:2513 Standard Microsystems Corp. 2.0 Hub
Bus 002 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
root@stm32mp1-av96:~# lsusb -t
/:  Bus 02.Port 1: Dev 1, Class=root_hub, Driver=ehci-platform/2p, 480M
    |__ Port 1: Dev 2, If 0, Class=Hub, Driver=hub/3p, 480M
        |__ Port 2: Dev 3, If 0, Class=Mass Storage, Driver=usb-storage, 480M
/:  Bus 01.Port 1: Dev 1, Class=root_hub, Driver=dwc2/1p, 480M
root@stm32mp1-av96:~# mount | grep sda
/dev/sda1 on /home/root/sda1 type vfat (rw,relatime,fmask=0022,dmask=0022,codepage=437,
↪iocharset=iso8859-1,shortname=mixed,errors=remount-ro)
```

USB OTG

The board supports that feature. For now it only works in DFU mode with STM32 Cube Programmer. Using the board as USB Gadget is currently under development.

eMMC

It can be used to store the firmware with STM32 Cube Programmer. It can also be mounted under Linux booted from another medium:

```
root@stm32mp1-av96:~# mount /dev/mmcblk2p4 emmc/
[ 3006.721643] EXT4-fs (mmcblk2p4): recovery complete
[ 3006.726627] EXT4-fs (mmcblk2p4): mounted filesystem with ordered data mode. Opts:␣
↪(null)
[ 3006.733931] ext4 filesystem being mounted at /home/root/emmc supports timestamps␣
↪until 2038 (0x7fffffff)
root@stm32mp1-av96:~# ls -l emmc
drwxr-xr-x    2 root     root          1024 Mar  9 12:34 bin
drwxr-xr-x    2 root     root          1024 Mar  9 12:34 boot
drwxr-xr-x    2 root     root          1024 Mar  9 12:34 dev
drwxr-xr-x   17 root     root          1024 Mar  9 12:34 etc
drwxr-xr-x    3 root     root          1024 Mar  9 12:34 home
drwxr-xr-x    3 root     root          1024 Mar  9 12:34 lib
drwx------    2 root     root         12288 Jan 12  2021 lost+found
drwxr-xr-x    2 root     root          1024 Mar  9 12:34 media
```

(continues on next page)

(continued from previous page)

```
drwxr-xr-x    2 root     root          1024 Mar  9 12:34 mnt
dr-xr-xr-x    2 root     root          1024 Mar  9 12:34 proc
drwxr-xr-x    2 root     root          1024 Jan  1  2000 run
drwxr-xr-x    2 root     root          1024 Mar  9 12:34 sbin
dr-xr-xr-x    2 root     root          1024 Mar  9 12:34 sys
lrwxrwxrwx    1 root     root             8 Mar  9 12:34 tmp -> /var/tmp
drwxr-xr-x   10 root     root          1024 Mar  9 12:34 usr
drwxr-xr-x    8 root     root          1024 Mar  9 12:34 var
```

Radio

Radio relies on proprietary BRCM firmware. It is already included in the image.

WiFi

WiFi can be controlled with `wpa_supplicant`, which is a standard Linux tool. Please refer to the tool manual for the details.

Example `wpa_suppliant` configs look like below. Assuming the config is saved in a file named `wpa.conf` and the interface is named `wlan0`, WiFi can be brought up with `wpa_supplicant -i wlan0 -c ./wpa.conf`:

```
# Access Point mode example configuration
fast_reauth=1
update_config=1

ap_scan=2
network={
        ssid="Avenger96 AP"
        mode=2
        frequency=2412
        key_mgmt=WPA-PSK
        proto=RSN
        pairwise=CCMP
        psk="PlaintextPasswordsAreGreat"
}
```

```
# Connection to an open network with broadcasted SSID
network={
        ssid="0xDEADBEEF"
        key_mgmt=NONE
}
```

For any fault in the hardware device, see *How to handle faulty hardware device*.

Bluetooth

Bluetooth be controlled with `bluetoothctl`, which is a standard Linux tool. Please refer to the tool manual for the details. Devices scanning can be enabled as follows:

```
root@stm32mp1-av96:~# bluetoothctl
Agent registered
[CHG] Controller 00:9D:6B:AA:77:68 Pairable: yes
[bluetooth]# power on
Changing power on succeeded
[CHG] Controller 00:9D:6B:AA:77:68 Powered: yes
[bluetooth]# discoverable on
Changing discoverable on succeeded
[CHG] Controller 00:9D:6B:AA:77:68 Discoverable: yes
[bluetooth]# scan on
Discovery started
[CHG] Controller 00:9D:6B:AA:77:68 Discovering: yes
[NEW] Device E2:A0:50:99:C9:61 Hue Lamp
[NEW] Device 57:2D:D5:48:8C:D0 57-2D-D5-48-8C-D0
[NEW] Device E4:04:39:65:9C:2A TomTom GPS Watch
[NEW] Device C0:28:8D:49:67:7E C0-28-8D-49-67-7E
```

Pairing and establishing connection is possible with `pair` and `connect` commands.

For any fault in the hardware device, see *How to handle faulty hardware device*.

96Boards Nitrogen

Contents

Overview

Nitrogen, a compliant IoT Edition board provides economical and compact BLE solutions for various IoT projects. This board includes the below features:

- Nordic nRF52832 microcontroller
- 64 KB of RAM
- 512 KB on-board flash storage.

Nitrogen hardware supports the Nordic Semiconductor nRF52832 ARM Cortex-M4F CPU.

Hardware

- For detailed specifications, see Nitrogen product page on the 96Boards website.
- For hardware user manual, see Seeed wiki.
- For hardware schematics, see Seeed Document.

For more details on 96Boards Nitrogen, see Nitrogen product page.

Working with the Board

Building an Application

Oniro Project OS Zephyr flavour is based on Zephyr kernel.

- Source the environment with proper template settings, flavour being zephyr and target machine being 96b-nitrogen:

```
$ TEMPLATECONF=../oniro/flavours/zephyr . ./oe-core/oe-init-build-env build-oniro-zephyr
```

- You will find yourself in the newly created build directory. Call bitbake to build the image. The supported image name is zephyr-philosophers.

```
$ MACHINE=96b-nitrogen bitbake zephyr-philosophers
```

MACHINE variable can be set up in conf/local.conf file under build directory or via command line.

Flashing an Application

Installing pyOCD

pyOCD is an open source Python package for programming and debugging Arm Cortex-M microcontrollers using multiple supported types of USB debug probes. It is fully cross-platform, with support for Linux.

- The latest stable version of pyOCD can be installed via pip as follows:

```
$ pip install --pre -U pyOCD
```

- To install the latest pre-release version from the HEAD of the master branch, do the following:

```
$ pip install --pre -U git+https://github.com/mbedmicro/pyOCD.git
```

- To install directly from the source by cloning the git repository, do the following:

```
$ python setup.py install
```

- Verify that the board is detected by pyOCD by executing the command:

```
$ pyocd-flashtool -l
```

Note: When *ValueError: The device has no langid* error is displayed due to lack of permission, perform the instructions as suggested in https://github.com/pyocd/pyOCD/tree/master/udev.

How to Flash

- To flash the image, execute the command used to build the image with -c flash_usb appended. For example, to flash the already built zephyr-philosophers image, do:

```
$ MACHINE=96b-nitrogen bitbake zephyr-philosophers -c flash_usb
```

SBC-B68-eNUC SECO

Contents

Overview

The SBC-B68-eNUC is a flexible and expandable full industrial x86 embedded NUC™ SBC with the Intel® Atom X Series, Intel® Celeron® J / N Series and Intel® Pentium® N Series (formerly code name Apollo Lake) Processors. Also available in industrial temperature version, the board offers wide range of connectivity options through WLAN and WWAN M.2 slots as well as wide input voltage range. Featuring Quad Channel soldered down LPDDR4-2400 memory, up to 8GB, thanks to its versatile expansion capabilities it is particularly suitable for embedded applications like HMI, multimedia devices, industrial IoT and industrial automation.

Hardware

For more detailed specifications of SBC-B68-eNUC SECO board, see SBC-B68-eNUC Specification.

Working with the Board

Building an Oniro image

To clone the source code, perform the procedure in: *Setting up a repo workspace*.

Linux image

1. Source the environment with proper template settings, flavour being *linux* and target machine being *seco-intel-b68*.

```
$ TEMPLATECONF=../oniro/flavours/linux . ./oe-core/oe-init-build-env build-oniro-linux
```

2. You will find yourself in the newly created build directory. Call *bitbake* to build the image. For example, if you are using *oniro-image-base* run the following command:

```
$ MACHINE=seco-intel-b68 bitbake oniro-image-base
```

To generate images for SSD Disk, refer to the following Flashing an Oniro image section.

Flashing an Oniro image

Linux image

USB Storage

Prerequisites

- Mini DisplayPort to HDMI converter cable
- HDMI Monitor
- USB Storage
- Linux Host

To flash Oniro image using USB storage, perform the following steps:

Prepare an Oniro Bootable USB Stick

1. Connect USB storage to your host PC.
2. After the image is built, you are ready to burn the generated image onto the USB storage. We recommend using *bmaptool* <*https://github.com/intel/bmap-tools*> and the instructions below will use it. For example, if you are building oniro-image-base run the following command replacing (or defining) `$DEVNODE` accordingly:

```
$ cd tmp/deploy/images/seco-intel-b68
$ bmaptool copy oniro-image-base-seco-intel-b68.wic.bz2 $DEVNODE
```

3. Put the card to the board and turn it on.

Run Oniro image

1. Connect bootable USB to target
2. Connect mini DP++ to HDMI adapter to HDMI monitor
3. Power on B68 and press **Esc** to enter **BIOS** mode.
4. Go to Save and Exit submenu
5. Select the bootable USB device under **Boot Override** and press Enter.

Testing the Board

Ethernet

Wired connection works out of the box. You can use standard tools like `ip`, `ifconfig` to configure the connection.

For any fault in the hardware device, see *How to handle faulty hardware device*.

USB Host

```
root@seco-intel-b68:~# lsusb
/:  Bus 02.Port 1: Dev 1, Class=root_hub, Driver=xhci_hcd/7p, 5000M
/:  Bus 01.Port 1: Dev 1, Class=root_hub, Driver=xhci_hcd/8p, 480M
```

eMMC

```
root@seco-intel-b68:~# fdisk -l /dev/mmcblk1
Disk /dev/mmcblk1: 29 GB, 31268536320 bytes, 61071360 sectors
954240 cylinders, 4 heads, 16 sectors/track
Units: sectors of 1 * 512 = 512 bytes
```

PCI Buses

```
root@seco-intel-b68:~# lspci
00:00.0 Host bridge: Intel Corporation Celeron N3350/Pentium N4200/Atom E3900 Series␣
↪Host Bridge (rev 0b)
00:02.0 VGA compatible controller: Intel Corporation HD Graphics 500 (rev 0b)
00:0e.0 Audio device: Intel Corporation Celeron N3350/Pentium N4200/Atom E3900 Series␣
↪Audio Cluster (rev 0b)
00:0f.0 Communication controller: Intel Corporation Celeron N3350/Pentium N4200/Atom␣
↪E3900 Series Trusted Execution Engine (rev 0b)
00:12.0 SATA controller: Intel Corporation Celeron N3350/Pentium N4200/Atom E3900 Series␣
↪SATA AHCI Controller (rev 0b)
00:13.0 PCI bridge: Intel Corporation Celeron N3350/Pentium N4200/Atom E3900 Series PCI␣
↪Express Port A #3 (rev fb)
00:13.3 PCI bridge: Intel Corporation Celeron N3350/Pentium N4200/Atom E3900 Series PCI␣
↪Express Port A #4 (rev fb)
00:15.0 USB controller: Intel Corporation Celeron N3350/Pentium N4200/Atom E3900 Series␣
↪USB xHCI (rev 0b)
```

(continues on next page)

(continued from previous page)

```
00:16.0 Signal processing controller: Intel Corporation Celeron N3350/Pentium N4200/Atom␣
↪E3900 Series I2C Controller #1 (rev 0b)
00:16.3 Signal processing controller: Intel Corporation Celeron N3350/Pentium N4200/Atom␣
↪E3900 Series I2C Controller #4 (rev 0b)
00:17.0 Signal processing controller: Intel Corporation Celeron N3350/Pentium N4200/Atom␣
↪E3900 Series I2C Controller #5 (rev 0b)
00:17.1 Signal processing controller: Intel Corporation Celeron N3350/Pentium N4200/Atom␣
↪E3900 Series I2C Controller #6 (rev 0b)
00:18.0 Signal processing controller: Intel Corporation Celeron N3350/Pentium N4200/Atom␣
↪E3900 Series HSUART Controller #1 (rev 0b)
00:18.2 Signal processing controller: Intel Corporation Celeron N3350/Pentium N4200/Atom␣
↪E3900 Series HSUART Controller #3 (rev 0b)
00:1b.0 SD Host controller: Intel Corporation Celeron N3350/Pentium N4200/Atom E3900␣
↪Series SDXC/MMC Host Controller (rev 0b)
00:1c.0 SD Host controller: Intel Corporation Celeron N3350/Pentium N4200/Atom E3900␣
↪Series eMMC Controller (rev 0b)
00:1f.0 ISA bridge: Intel Corporation Celeron N3350/Pentium N4200/Atom E3900 Series Low␣
↪Pin Count Interface (rev 0b)
00:1f.1 SMBus: Intel Corporation Celeron N3350/Pentium N4200/Atom E3900 Series SMBus␣
↪Controller (rev 0b)
01:00.0 Ethernet controller: Intel Corporation I210 Gigabit Network Connection (rev 03)
02:00.0 Ethernet controller: Intel Corporation I210 Gigabit Network Connection (rev 03)
```

Loaded Modules

```
root@seco-intel-b68:~# lsmod
Module                  Size  Used by
nfc                    73728  0
bnep                   20480  2
uio                    20480  0
snd_hda_codec_hdmi     53248  1
iwlwifi               299008  0
cfg80211              688128  1 iwlwifi
snd_hda_codec_cirrus    20480  1
snd_hda_codec_generic    65536  1 snd_hda_codec_cirrus
ledtrig_audio          16384  1 snd_hda_codec_generic
intel_rapl_msr         16384  0
snd_soc_skl           114688  0
snd_soc_sst_ipc        16384  1 snd_soc_skl
snd_soc_sst_dsp        24576  1 snd_soc_skl
snd_hda_ext_core       20480  1 snd_soc_skl
snd_soc_acpi_intel_match    36864  1 snd_soc_skl
snd_soc_acpi           16384  2 snd_soc_acpi_intel_match,snd_soc_skl
snd_soc_core          200704  1 snd_soc_skl
intel_rapl_common      20480  1 intel_rapl_msr
snd_compress           20480  1 snd_soc_core
ac97_bus               16384  1 snd_soc_core
intel_pmc_bxt          16384  0
intel_telemetry_pltdrv    20480  0
intel_telemetry_core    16384  1 intel_telemetry_pltdrv
```

(continues on next page)

(continued from previous page)

```
snd_hda_intel          32768  0
x86_pkg_temp_thermal    16384  0
snd_intel_dspcfg       16384  2 snd_hda_intel,snd_soc_skl
snd_hda_codec          98304  4 snd_hda_codec_generic,snd_hda_codec_hdmi,snd_hda_intel,
↪snd_hda_codec_cirrus
coretemp               16384  0
snd_hda_core           65536  7 snd_hda_codec_generic,snd_hda_codec_hdmi,snd_hda_intel,
↪snd_hda_ext_core,snd_hda_codec,snd_hda_codec_cirrus,snd_soc_skl
snd_pcm                86016  7 snd_hda_codec_hdmi,snd_hda_intel,snd_hda_codec,snd_
↪compress,snd_soc_core,snd_soc_skl,snd_hda_core
snd_timer              32768  1 snd_pcm
i915                 1888256  5
mei_me                 32768  0
video                  40960  1 i915
mei                    81920  1 mei_me
```

Video

Output video tested with *DP++* to *HDMI* adapter.

SBC-C61 SECO

Contents

Overview

SBC-C61 is an SBC built upon the NXP i.MX 8M mini Application Processors characterised by HEVC/VP9 decoding in 1080p60. As for the memory, it features a LPDDR4 RAM. The range of connectivity options is particularly broad, with optional Wi-Fi and BT LE 4.2 and optionally soldered on-board LTE Cat 4 Modem with microSIM slot or eSIM. Interestingly, it also features a Cortex-M4, that is real-time operating system capable for serving real-time applications that process data as it comes in without buffer delays.

Hardware

For more detailed specifications of SBC-C61 SECO board, see SBC-C61 Specification.

Working with the Board

Building an Oniro image

To clone the source code, perform the procedure in: *Setting up a repo workspace*.

Linux image

1. Source the environment with proper template settings, flavour being `linux` and target machine being `seco-imx8mm-c61`.

```
$ TEMPLATECONF=../oniro/flavours/linux . ./oe-core/oe-init-build-env build-oniro-linux
```

2. You will find yourself in the newly created build directory. Call `bitbake` to build the image. The supported image is `oniro-image-base`.

```
$ MACHINE=seco-imx8mm-c61 bitbake oniro-image-base
```

To generate images for eMMC, refer to the following flashing procedure.

Flashing an Oniro image

Linux image

MMC Storage

Prerequisites

- USB To UART adapter
- USB to OTG adapter
- Download and install mfgtools
- Linux Host

To flash Oniro image using USB to OTG adapter, perform the following steps:

1. Short circuit pin 1 and 2 of CN52 pin header to enter the Serial Download mode.
2. Connect USB to OTG adapter to your host PC

3. Navigate to the inside build output directory:

```
$ cd tmp/deploy/images/seco-imx8mm-c61/
```

4. Unzip build output using Gzip software:

```
$ gzip -d oniro-image-base-seco-imx8mm-c61.wic.gz
```

5. To write uboot and image(p1:kernel, p2:dtb, rootfs) into c61 mmc via mfgtools:

```
$ sudo uuu -b emmc_all imx-boot-seco-imx8mm-c61-emmc.bin-flash_evk oniro-image-base-
↪seco-imx8mm-c61.wic
```

6. Power ON SBC-C61
7. Remove **CN52 short circuit**
8. Press the reset button

Testing the Board

Ethernet

You can use standard tools like `ip`, `ifconfig` to configure the connection.

```
root@seco-imx8mm-c61:~# ifconfig
eth0      Link encap:Ethernet  HWaddr 1A:20:58:83:70:F0
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

For any fault in the hardware device, see *How to handle faulty hardware device*.

USB Host

```
root@seco-imx8mm-c61:~# lsusb
Bus 001 Device 003: ID 058f:6387 Alcor Micro Corp. Flash Drive
Bus 001 Device 002: ID 0424:2514 Standard Microsystems Corp. USB 2.0 Hub
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

eMMC

```
root@seco-imx8mm-c61:~# fdisk -l /dev/mmcblk0
Disk /dev/mmcblk0: 59 GB, 63585648640 bytes, 124190720 sectors
1940480 cylinders, 4 heads, 16 sectors/track
Units: sectors of 1 * 512 = 512 bytes

Device         Boot StartCHS    EndCHS         StartLBA     EndLBA    Sectors  Size Id Type
```

(continues on next page)

(continued from previous page)

```
/dev/mmcblk0p1 *  64,0,1      893,3,4          8192      114403      106212 51.8M  c␣
↪Win95 FAT32 (LBA)
/dev/mmcblk0p2    896,0,1     1023,3,32      114688      558903      444216  216M 83 Linux
```

Loaded Modules

```
root@seco-imx8mm-c61:~# lsmod
Module                  Size  Used by
nfc                    90112  0
bluetooth             409600  8
ecdh_generic           16384  1 bluetooth
ecc                    32768  1 ecdh_generic
rfkill                 36864  3 nfc,bluetooth
ipv6                  442368  26
caam_jr               196608  0
caamhash_desc          16384  1 caam_jr
caamalg_desc           36864  1 caam_jr
crypto_engine          16384  1 caam_jr
rng_core               24576  1 caam_jr
authenc                16384  1 caam_jr
libdes                 24576  1 caam_jr
snd_soc_simple_card    20480  0
fsl_imx8_ddr_perf      20480  0
crct10dif_ce           20480  1
snd_soc_simple_card_utils    24576  1 snd_soc_simple_card
rtc_snvs               16384  1
snvs_pwrkey            16384  0
caam                   40960  1 caam_jr
clk_bd718x7            16384  0
error                  24576  4 caamalg_desc,caamhash_desc,caam,caam_jr
imx8mm_thermal         16384  0
snd_soc_fsl_sai        20480  0
imx_cpufreq_dt         16384  0
```

Raspberry Pi 4 Model B

Contents

Overview

Raspberry Pi 4 Model B is powered with Broadcom BCM2711, quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz. This product's key features include a high-performance 64-bit quad-core processor, dual-display support at resolutions up to 4K via a pair of micro-HDMI ports, hardware video decode at up to 4Kp60, and the RAM size various from 2GB, 4GB, or 8GB, dual-band 2.4/5.0GHz wireless LAN, Bluetooth 5.0, Gigabit Ethernet, USB 3.0, and PoE capability (via a separate PoE HAT add-on). The dual-band wireless LAN and Bluetooth have modular compliance certification, allowing the board to be designed into end products with significantly reduced compliance testing, improving both cost and time to market.

Applications

- Embedded Design & Development
- Hobby & Education
- IoT (Internet of Things)
- Communications & Networking

Hardware

- For Raspberry Pi 4 Model B Schematics, see RaspberryPi official website.
- For Raspberry Pi 4 Model B datasheet, see RaspberryPi official website.
- For Raspberry Pi 4 boot EEPROM, see RaspberryPi official website.

For more details on the Raspberry Pi 4 board, see Raspberry Pi hardware page.

Working with the board

Building Oniro Project image

To clone the source code, perform the procedure in: *Setting up a repo workspace*.

Linux image

1. Source the environment with proper template settings, the flavour being *linux* and target machine being *raspberrypi4-64*. Pay attention to how relative paths are constructed. The value of *TEMPLATECONF* is relative to the location of the build directory *./build-linux-raspberrypi4-64*, which is going to be created after this step:

```
$ TEMPLATECONF=../oniro/flavours/linux . \
  ./oe-core/oe-init-build-env build-oniro-linux-raspberrypi4-64
```

2. You will find yourself in the newly created build directory. Call *bitbake* to build the image. For example, if you are using *oniro-image-base* run the following command:

```
$ MACHINE=raspberrypi4-64 bitbake oniro-image-base
```

3. After the build completes, the bootloader, kernel, and rootfs image files can be found in `build-oniro-linux-raspberrypi4-64/tmp/deploy/images/$MACHINE/`. The key file which is needed to flash into the SD card is `oniro-image-base-raspberrypi4-64.wic.bz2`.

Flashing Oniro Project Linux image

SD Card

The Raspberry Pi 4 board support multiple boot options. The below section describes booting the board with an SD card option.

1. After the image is built, you are ready to burn the generated image onto the SD card. We recommend using *bmaptool <https://github.com/intel/bmap-tools>* and the instructions below will use it. For example, if you are building oniro-image-base run the following command replacing (or defining) `$DEVNODE` accordingly:

```
$ cd tmp/deploy/images/raspberrypi4-64
$ bmaptool copy oniro-image-base-raspberrypi4-64.wic.bz2 $DEVNODE
```

2. Put the card to the board and turn it on.

Testing the Board

HDMI

Two micro HDMI ports (HDMI-0 and HDMI-1) are enabled by default. Simply plugging your HDMI-equipped monitor into the RPi4 using a standard HDMI cable will automatically lead to the Pi using the best resolution the monitor supports.

For more details, see HDMI ports and configuration.

Bluetooth & BLE

By default, BT and BLE are supported.

For any fault in the hardware device, see *How to handle faulty hardware device*.

Ethernet & WiFi

Drivers for both Ethernet and WiFi is available by default and hence no specific configuration is needed to enable drivers for these interfaces.

Setting a static of dynamic IP for the interface is implementation and deployment specific and any network configuration tool can be used to configure IPv4 or IPv6 address to RPi.

To set up Wi-Fi connection, use `nmcli` by executing the following command:

```
# nmcli dev wifi connect "network-ssid" password "network-password"
```

For any fault in the hardware device, see *How to handle faulty hardware device*.

Audio

To enable the audio over 3.5mm jack, add the following line in your build's `local.conf`:

```
RPI_EXTRA_CONFIG = "dtparam=audio=on"
```

To enable the `aplay` support for audio playback, append the following lines:

```
IMAGE_INSTALL_append = " gstreamer1.0  gstreamer1.0-meta-base
gstreamer1.0-plugins-base gstreamer1.0-plugins-good"
IMAGE_INSTALL_append = " alsa-lib alsa-utils alsa-tools"
```

To test the audio out on the *3.5mm audio jack*, we need to download the wav file and play with `aplay`.

```
# wget https://file-examples-com.github.io/uploads/2017/11/file_example_WAV_1MG.wav
# aplay file_example_WAV_1MG.wav
```

Connect the headset on *3.5mm audio jack* and you should be able to hear the audio.

I2C

I2C is disabled by default. To enable I2C, edit the `local.conf` build's configuration adding:

```
ENABLE_I2C = "1"
```

The device tree does not create the I2C devices. For a quick test, install the module.

```
root@raspberrypi4-64:~# modprobe i2c_dev
[  611.019250] i2c /dev entries driver

root@raspberrypi4-64:~# ls -ls /dev/i2c-1
    0 crw-------    1 root     root       89,   1 Mar 29 10:41 /dev/i2c-1
```

Note: Need to be updated with more options.

GPIO

GPIO testing can be done using the sysfs Interface.

The following example shows how to test the GPIO-24 (which corresponds to physical pin number 18 on the GPIO connector of the Raspberry Pi):

By default, sysfs driver is loaded, you will see the GPIO hardware exposed in the file system under `/sys/class/gpio`. It might look something like this:

```
root@raspberrypi4-64:/sys/class/gpio# ls /sys/class/gpio/
export       gpiochip0    gpiochip504  unexport
```

We'll look at how to use this interface next. Note that the device names starting with `gpiochip` are the GPIO controllers and we won't directly use them.

To use a GPIO pin from the sysfs interface, perform the following steps:

1) Export the pin.

```
# echo 24 >/sys/class/gpio/export
```

2) Set the pin direction (input or output).

```
# echo out >/sys/class/gpio/gpio24/direction
```

3) If an output pin, set the level to low or high.

To validate the GPIO24 pin value, connect the LED light with the positive line on pin #18 (GPIO24) and the negative line on pin #20 (Ground).

```
# echo 0 >/sys/class/gpio/gpio24/value  # to set it low - LED Turn OFF
# echo 1 >/sys/class/gpio/gpio24/value  # to set it high - LED Turn ON
```

4) If an input pin, read the pin's level (low or high).

```
# cat /sys/class/gpio/gpio24/value  # 0 is low & 1 is high.
```

5) When done, unexport the pin.

```
# echo 24 >/sys/class/gpio/unexport
```

Arduino Nano 33 BLE

Contents

Overview

The Arduino Nano 33 BLE Sense is Arduino's 3.3V AI-enabled board in the smallest available form factor: 45x18mm!

The Arduino Nano 33 BLE Sense is an entirely new board on a well-known form factor. It comes with a series of embedded sensors:

- 9 axis inertial sensor: This makes this board ideal for wearable devices.
- Humidity and temperature sensor: To get highly accurate measurements of the environmental conditions.
- Barometric sensor: Make a simple weather station.
- Microphone: To capture and analyze sound in real-time.
- Gesture, proximity, light color, and light intensity sensor: Estimate the room's luminosity, but also whether someone is moving close to the board.

Arduino Nano 33 BLE hardware supports the Nordic Semiconductor nRF52840 ARM Cortex-M4 CPU running at 64 MHz.

Hardware

- For detailed specifications, see Arduino Nano 33 BLE product page on the Arduino website.
- For product specification and Datasheet, see Arduino page.
- For hardware schematics, see Arduino.

Working with the Board

Building an Application

Oniro Project Zephyr flavour is based on the Zephyr kernel.

1. Source the environment with proper template settings, the flavour being *zephyr* and target machine being *arduino-nano-33-ble*:

```
$ TEMPLATECONF=../oniro/flavours/zephyr . ./oe-core/oe-init-build-env build-
↪oniro-zephyr
```

2. You will find yourself in the newly created build directory. Call *bitbake* to build the image. The supported image name is *zephyr-philosophers*.

```
$ MACHINE=arduino-nano-33-ble bitbake zephyr-philosophers
```

You can set-up MACHINE variable in `conf/local.conf` file under the build directory, or via the command line.

Alternatively you might want to build the Arduino's blinking LED sample application, *blinky*. In order to do so issue the following:

```
$ MACHINE=arduino-nano-33-ble bitbake zephyr-blinky
```

3. After the build completes, the `zephyr-philosophers.bin` and the `zephyr-blinky.bin` file can be found in `build-oniro-zephyr/tmp-newlib/deploy/images/arduino-nano-33-ble/`.

Flashing an Application

1. Enable the permissions for board connected port:

```
$ sudo usermod -a -G dialout `whoami`
$ sudo chmod a+rw /dev/ttyACM0
```

2. To flash the image, execute the command used to build the image with -c flash_usb appended. For example, to flash the already built zephyr-philosophers image, execute:

```
$ MACHINE=arduino-nano-33-ble bitbake zephyr-philosophers -c flash_usb
```

3. Run your favorite terminal program to listen for output.

```
$ minicom -D /dev/ttyACM0
```

Configure the connection as follows:

- Baud Rate: 115200
- Data: 8 bits
- Parity: None
- Stop bits: 1

Note: If no output is displayed, set the permissions again as mentioned in Step-1 of of this section.

4. Firmware output can be viewed in minicom with:

```
Philosopher 5 [C:-2]        STARVING
Philosopher 3 [P: 0]    DROPPED ONE FORK
Philosopher 3 [P: 0]  THINKING [  25 ms ]
Philosopher 2 [P: 1]   EATING  [  425 ms ]
Philosopher 3 [P: 0]        STARVING
Philosopher 4 [C:-1]        STARVING
Philosopher 4 [C:-1]    HOLDING ONE FORK
Philosopher 4 [C:-1]   EATING  [  800 ms ]
Philosopher 3 [P: 0]    HOLDING ONE FORK
Philosopher 2 [P: 1]    DROPPED ONE FORK
Philosopher 2 [P: 1]  THINKING [  725 ms ]
Philosopher 1 [P: 2]   EATING  [  225 ms ]
```

nRF52840 DK

Contents

Overview

The nRF52840 DK is a low-cost single-board development kit that uses the nRF52840 multi-protocol SoC to develop Bluetooth® 5, Bluetooth mesh, Thread, Zigbee, ANT, IEEE 802.15.4, and 2.4 GHz proprietary applications. It also supports development on the nRF52811 SoC.

Hardware

- For detailed specifications, see nRF52840 DK product page on the nRF52840 DK website.
- For hardware schematics, see nRF52840 Development Kit.

Working with the Board

Building an Application

Oniro Project Zephyr flavour is based on Zephyr kernel.

1. Source the environment with proper template settings, flavour being `zephyr` and target machine being `nrf52840dk-nrf52840`:

   ```
   $ TEMPLATECONF=../oniro/flavours/zephyr . ./oe-core/oe-init-build-env build-
   ↪oniro-zephyr
   ```

2. You will find yourself in the newly created build directory. Call `bitbake` to build the image. Example below shows how to build `zephyr-philosophers`.

   ```
   $ MACHINE=nrf52840dk-nrf52840 bitbake zephyr-philosophers
   ```

   You can set up `MACHINE` variable can be set up in `conf/local.conf` file under the build directory, or via the command line.

3. After the build completes, the image file can be found in `build-oniro-zephyr/tmp-newlib/deploy/images/nrf52840dk-nrf52840/`.

Flashing an Application

pyOCD is a required host tool used by the flashing mechanism described below:

- To install the latest stable version of pyOCD via pip as follows:

```
$ pip install --pre -U pyOCD
```

Note: When `ValueError: The device has no langid` error is displayed due to lack of permission, perform the instructions to resolve.

- To flash the image, execute the command used to build the image with `-c flash_usb` appended. For example, to flash the already built zephyr-philosophers image, execute:

```
$ MACHINE=nrf52840dk-nrf52840 bitbake zephyr-philosophers -c flash_usb
```

2.9.2 Supported Virtual Targets

This section details the support for virtual targets in Oniro Project.

Qemu X86-64

Contents

Overview

Oniro Project supports running the software stack into an virtual environment using Qemu.

Building an Oniro image

To clone the source code, perform the procedure in: *Setting up a repo workspace*.

Building a Linux image

Build Steps

1. Source the environment with proper template settings, flavour being *linux* and target machine being *qemux86-64*. Pay attention to how relative paths are constructed. The value of *TEMPLATECONF* is relative to the location of the build directory *./build-oniro-linux*, that is going to be created after this step:

```
$ TEMPLATECONF=../oniro/flavours/linux . ./oe-core/oe-init-build-env build-oniro-linux
```

2. You will find yourself in the newly created build directory. Call *bitbake* to build the image. For example, if you are using *oniro-image-base* run the following command:

```
$ MACHINE=qemux86-64 bitbake oniro-image-base
```

Once the image is done, you can run the Qemu using the provided script wrapper:

```
$ MACHINE=qemux86-64 runqemu oniro-image-base wic
```

Qemu X86

Contents

Overview

Oniro Project supports running the software stack into an virtual environment using Qemu.

Building an Oniro image

To clone the source code, perform the procedure in: *Setting up a repo workspace*.

Building a Linux image

Build Steps

1. Source the environment with proper template settings, flavour being *linux* and target machine being *qemux86*. Pay attention to how relative paths are constructed. The value of *TEMPLATECONF* is relative to the location of the build directory *./build-oniro-linux*, that is going to be created after this step:

```
$ TEMPLATECONF=../oniro/flavours/linux . ./oe-core/oe-init-build-env build-oniro-linux
```

2. You will find yourself in the newly created build directory. Call *bitbake* to build the image. For example, if you are using *oniro-image-base* run the following command:

```
$ MACHINE=qemux86 bitbake oniro-image-base
```

Once the image is done, you can run the Qemu using the provided script wrapper:

```
$ MACHINE=qemux86 runqemu oniro-image-base wic
```

2.9.3 Adding New Hardware Support in Oniro Project

This section details the addition of new hardware to the supported set in Oniro Project. It is intended as a checklist for adding new boards to Oniro Project build system.

Before starting get familiar with Oniro Project Contribution Process.

Contents

Select Oniro Project Flavour

Oniro Project uses a notion of kernel specific flavours:

- Linux flavour
- Zephyr flavour
- FreeRTOS flavour (experimental)

Flavours have predefined *IMAGES* and *MACHINES*.

A single board can be included in more than one flavour only when it has **well maintained support** in targeted kernels.

Add Required meta-layers

Oniro flavours configuration templates (stored in distro/oniro/flavours directory) consist of the following files:

Table 4: Configuration Files

File Name	Description
bblayers.conf.sample	set of meta-layers for the specific flavour (it can be unified across multiple layers where there are no layers incompatibilities)
conf-notes.txt	text snippet to be used as part of build logs
local.conf.sample	default flavour build configuration

Oniro Project build system uses ***repo*** tool for cloning required meta-layers into appropriate build directory structure (see *Setting up a repo workspace*). To include a new layer, it has to be added in two places of the oniro repository:

- The manifest file
- The flavours `bblayers.conf.sample` file

Test Image Backward Compatibility Of Newly Added Layers

New BSP layers cannot interfere / break already supported `IMAGES` / `MACHINES`.

Document and Advertise the New `MACHINE` Support

Newly added `MACHINE` shall be documented in: *Hardware Support*. Use an existing board documentation as template and populate it accordingly for your newly added machine.

The same machine needs to also be advertised in two places:

- Flavour's `local.conf.sample` as a commented out `MACHINE` variable value (tweak this step accordingly for default machine change)
- Flavour's `conf-notes.txt` to surface the support in build logs

Create Merge Requests

Create the Merge Request against the ***dunfell*** branch according to the Contributing Process for repositories:

- distro/oniro

2.9.4 Oniro Project Support for Peripherals

This section details the supported peripherals as part of Oniro Project.

Supported WiFi Chipsets

Oniro Project provides support for the on-board WiFi chipsets on the targets it maintains compatibility with. For more info check the documentation for each *supported board*.

2.10 OpenThread

OpenThread is an open-source implementation of the Thread networking protocol based on IPv6. It is designed for low-powered devices that operate in a mesh network based on the IEEE 802.15.4 standards. Some of the main advantages of implementing OpenThread are:

- Simple to install and operate.
- Mandatory authentication of devices.
- All communications are encrypted.
- Self-healing mechanism with no single point of failure in a mesh network.
- Support for low-powered devices.
- Scalable up to hundreds of devices.

For more details on OpenThread, Click.

2.10.1 Connecting to Internet

You can connect the Thread network to the internet using the Thread Border Router. The Thread Border Router is an open-source implementation provided by the OpenThread community. For instructions on how to set up a Border Router, Click.

2.11 How to Handle Faulty Hardware Device?

In a situation where you have enabled a new board and one of the devices is faulty or has some issues (e.g. driver, hardware, firmware, and so on), to continue with the setup, it is important to have an alternative option for the faulty device.

You can replace the respective faulty device with an external devices listed in the following table. The devices listed are hardware components with opensource drivers and have no usage restrictions.

Table 5: Supported External Hardware Device

Device	Chip	Firmware status	Dongle name	Remarks
Ethernet	RTL8153	linux-firmware	Many available	SFP variants available
WiFi	RTl8192cu	linux-firmware, license	OURLiNK	Works out-of-the-box
WiFi	Ralink RT5372	linux-firmware, license	PAU05 - Panda Wireless	Works out-of-the-box
Bluetooth	BCM20702A0	linux-firmwar, license	PLUGABLE	Works out-of-the-box
Bluetooth	BCM20702A0	linux-firmwar, license	Kinivo	Works out-of-the-box
IEEE 802.15.4	AT86RF231	Flashed on device	ATUSB	Open hardware and firmware

CHAPTER

THREE

CONTRIBUTING TO ONIRO PROJECT

The contributing process and guidelines part of this document must be applied to any repository in the scope of the Oniro Project project. Each repository must include this information in its `CONTRIBUTING.md` file and optionally, complement the process and guidelines with repository-specific requirements.

3.1 Gitlab Contributions

Contents

3.1.1 Overview

Oniro Project project handles contributions as merge requests to relevant repositories part of the Oniro Project GitLab instance. The flow for handling that is classic: fork-based merge requests. This means that once you have an account, you can fork any repository, create a branch with proposed changes and raise a merge request against the forked repository. More generic information you can find on the Gitlab's documentation as part of "Merge requests workflow".

3.1.2 Commit Guidelines

Note: If you are new to `git`, start by reading the official Getting Started Document.

At its core, contributing to the Oniro Project project means *wrapping* your work as `git` commits. How we handle this has an impact on rebasing, cherry-picking, back-porting, and ultimately exposing consistent documentation through its logs.

To achieve this, we maintain the following commit guidelines:

- Each commit should be able to stand by itself providing a building block as part of the MR.
  - A good balance of granularity with scoped commits helps to handle backports (e.g. cherry-picks) and also improves the ability to review smaller chunks of code taking commit by commit.

- Changes that were added on top of changes introduced in the MR, should be squashed into the initial commit.
  - For example, a MR that introduced a new build system recipe and, as a separate commit, fixed a build error in the initial recipe. The latter commit should be squashed into the initial commit.
  - For example, a MR introducing a new docs chapter and also adding, as a separate commit, some typo fixes. The latter commits should be squashed into the initial commit.
  - There is a small set of exceptions to this rule. All these exceptions gravitate around the case where an MR, even if it provides multiple commits in the same scope (for example, to the same build recipe), each of the commits has a very specific purpose.
    - For example, a line formating change followed by a chapter addition change in the same documentation file.
    - Also, it can be the case of two functional changes that are building blocks in the same scope.
    - Another example where commits are not to be squashed is when having a commit moving the code and a commit modifying the code in the new location.
- Make sure you clean your code of trailing white spaces/tabs and that each file ends with a new line.
- Avoid *merge* commits as part of your MR. Your commits should be rebased on top of the *HEAD* of the destination branch.

As mentioned above, *git log* becomes informally part of the documentation of the product. Maintaining consistency in its format and content improves debugging, auditing, and general code browsing. To achieve this, we also require the following commit message guidelines:

- The *subject* line (the first line) needs to have the following format: `scope: Title limited to 80 characters`.
  - Use the imperative mood in the *subject* line for the *title*.
  - The *scope* prefix (including the colon and the following whitespace) is optional but most of the time highly recommended. For example, fixing an issue for a specific build recipe, would use the recipe name as the *scope*.
  - The *title* (the part after the *scope*) starts with a capital letter.
  - The entire *subject* line shouldn't exceed 80 characters (same text wrapping rule for the commit body).
- The commit *body* separated by an empty line from the *subject* line.
  - The commit *body* is optional but highly recommended. Provide a clear, descriptive text block that accounts for all the changes introduced by a specific commit.
  - The commit *body* must not contain more than 80 characters per line.
- The commit message will have the commit message *trailers* separated by a new line from the *body*.
  - Each commit requires at least a *Signed-off-by* trailer line. See more as part of the *DCO sign-off* document.
  - All *trailer* lines are to be provided as part of the same text block - no empty lines in between the *trailers*.

Additional commit message notes:

- Avoid using special characters anywhere in the commit message.
- Be succinct but descriptive.
- Have at least one *trailer* as part of each commit: *Signed-off-by*.
- You can automatically let `git` add the *Signed-off-by* by taking advantage of its `-s` argument.
- Whenever in doubt, check the existing log on the file (`<FILE>`) you are about to commit changes, using something similar to: `git log <FILE>`.

Example of a full git message:

```
busybox: Add missing dependency on virtual/crypt

Since version 1.29.2, the busybox package requires virtual/crypt. Add this
to DEPENDS to make sure the build dependency is satisfied.

Signed-off-by: Joe Developer <joe.developer@example.com>
```

3.1.3 Contributions to Documentation

In Oniro Project, the documentation usually stays with the respective code repositories. This means that contributing to documentation is not in any way different than contributing to code. The processes, contribution guidelines are to remain the same. The only difference is that documentation files are to be released under `Creative Commons License version 4.0`.

Documentation that doesn't link directly to one specific repository, is available in the docs repository.

In terms of file format, the project unifies its documentation as `ReStructuredText` files. A RestructuredText primer is available as part of the Sphinx documentation.

As a rule of thumb, anything that ends up compiled in the project documentation is to maintain the RestructuredText file format. Text files that are not meant to be compiled as part of the project's documentation can be written in Markdown. For example, a repository `README` file can be written in Markdown as it doesn't end up compiled in the project-wide documentation.

3.2 REUSE Compliance

Contents

3.2.1 SPDX Information and REUSE Standard

All projects and files for an hosted project **MUST** be REUSE compliant. REUSE requires SPDX information for each file, rules for which are as follows:

- Any new file must have a SPDX header (copyright and license).
- For files that don't support headers (for example binaries, patches etc.) an associated `.license` file must be included with the relevant SPDX information.
- Do not add Copyright Year as part of the SPDX header information.
- The general rule of thumb for the license of a patch file is to use the license of the component for which the patch applies.

- When modifying a file through this contribution process, you may (but don't have to) claim copyright by adding a copyright line.
- Never alter copyright statements made by others, but only add your own.

Some files will make an exception to the above rules as described below:

- Files for which copyright is not claimed and for which this information was not trivial to fetch (for example backporting patches, importing build recipes etc. when upstream doesn't provide the SPDX information in the first place)
- license files (for example `common-licenses` in bitbake layers)

SPDX Header Example

Make sure all of your submitted new files have a licensing statement in the headers. Please make sure that the license for your file is consistent with the licensing choice at project level and that you select the correct SPDX identifier, as in the following example for Apache 2.0 license:

```
/*
 * SPDX-FileCopyrightText: Jane Doe <jane@example.com>
 *
 * SPDX-License-Identifier: Apache-2.0
 */
```

Substantial Contributions

Therefore, if your contribution is only a patch directly applied to an existing file, then you are not required to do anything. If your contribution is an entire new project, or a substantial, copyrighted contribution, you **MUST** make sure that you do that following the IP Policy and that you comply with REUSE standard to include the licensing information where they are required.

3.3 DCO sign-off

Contents

3.3.1 Overview

Commits **MUST** be submitted only with a sign-off by the submitter. A commit without a sign-off will be automatically rejected. You don't need be the author or the copyright holder of the contribution, but you must make sure that you have the power to submit on behalf of those who are.

To sign your work, just add a line like this at the end of your commit message:

```
Signed-off-by: Jane Doe <jane@example.com>
```

This could be done automatically in the `git` submission:

```
git commit --signoff -m "comment"
```

3.3.2 Developer Certificate of Origin

By doing this you state that you certify the following (from https://developercertificate.org):

```
Developer Certificate of Origin
Version 1.1

Copyright (C) 2004, 2006 The Linux Foundation and its contributors.
1 Letterman Drive
Suite D4700
San Francisco, CA, 94129

Everyone is permitted to copy and distribute verbatim copies of this
license document, but changing it is not allowed.


Developer's Certificate of Origin 1.1

By making a contribution to this project, I certify that:

(a) The contribution was created in whole or in part by me and I
    have the right to submit it under the open source license
    indicated in the file; or

(b) The contribution is based upon previous work that, to the best
    of my knowledge, is covered under an appropriate open source
    license and I have the right under that license to submit that
    work with modifications, whether created in whole or in part
    by me, under the same open source license (unless I am
    permitted to submit under a different license), as indicated
    in the file; or

(c) The contribution was provided directly to me by some other
    person who certified (a), (b) or (c) and I have not modified
    it.

(d) I understand and agree that this project and the contribution
    are public and that a record of the contribution (including all
    personal information I submit with it, including my sign-off) is
```

(continues on next page)

(continued from previous page)

```
    maintained indefinitely and may be redistributed consistent with
    this project or the open source license(s) involved.
```

3.4 Contributing to Projects not Maintained by Oniro Project Team

3.4.1 Overview

In order to comply with *Upstream first* rule and Open Source licenses requirements, Oniro Project developers collaborate with several upstream projects to submit fixes, improvements, bug reports, problem investigation results etc. Contribution must be made in accordance with upstream project policy using the tooling upstream project prefers such as mailing list, github/gitlab pull/merge requests, etc.

3.4.2 Signing-off Contribution

All contributions must be signed off by the Oniro Project developer using their email account associated with the copyright owner of the work (in most cases it will be the corporate email address). This does not apply if the upstream project policy says otherwise or signing off of the contribution is not possible due to upstream project's limitation. It is recommended to use corporate email address as a sender address in case of email communication.

In case the Oniro Project developer contributes code written by someone else (provided by partner, end user, third-party contributor etc) original author's copyright must be kept and entire contribution must be signed off with "Author:" tag unless the author explicitly asks otherwise. This could be done in the `git` submission:

```
git commit --signoff --author="Foo Bar <foo.bar@example.com>" -m "comment"
```

By doing this Oniro Project developer states that they agree to the terms of *DCO*

The developer must make sure that they have rights to submit on behalf of the original author according to the license and/or author's permission.

It is Oniro Project developer's responsibility to check license compatibility between the contribution and the upstream project.

3.4.3 Contribution Agreement

In case the upstream project requires signing of contribution agreement of any kind, the Oniro Project developer must review it carefully before submitting the contribution. In case of any doubt they must contact their manager or legal team for further guidance.

3.4.4 Security-related Contribution and Sensitive Data

It is the Oniro Project developer's responsibility to verify the data they share with upstream counterpart to prevent leak of sensitive information. Special attention must be given in the case of security issues or issues which can be potentially rated as security-related in the future. Such cases must be handled separately according to upstream policy (using private channels or directly with the Security Officer if upstream has one).

3.5 Devtool

Contents

Devtool is a tool available on OpenEmbedded that allows you to start development with your OpenEmbedded distribution. This command tool is available in addition to the bitbake command. The *devtool* command is an important component of the SDK's extensibility. Run `devtool --help` to view the devtool help commands.

This tool commands allows to:

- devtool add: Assists in the development of a new recipe to build a specified source tree.
- devtool modify: Sets up the build environment to modify the source for an existing recipe.
- devtool upgrade: Upgrades an existing recipe to a new upstream version.

For more information on using devtool in your sdk workflow, see Use the Extensible SDK.

3.5.1 Adding a New Recipe

To add a new recipe *busybox* to the workspace layer, perform the following procedure:

1. Add a new recipe *busybox* to the workspace to build a specified source tree, execute:

   ```
   $ devtool add busybox mysources/busybox
   ```

   The above example creates and adds a new recipe named *busybox* to the workspace layer.

   Note: The `devtool add` command creates the workspace layer when you add a recipe and the workspace layer does not exist.

2. Edit the source code file and commit the changes to your workspace, execute:

   ```
   $ devtool edit-recipe busybox
   ```

   Executing the above command opens the default editor (as specified by the editor variable) on the specified recipe.

3. Build your recipe *busybox* from the workspace, execute:

   ```
   $ devtool build busybox
   ```

4. Deploy the recipe *busybox* build output to test on the live target machine, execute:

   ```
   $ devtool deploy-target busybox root@<ip of board>
   ```

5. Populate the workspace layer with your new recipe in the *<WORKSPACE_LAYER_PATH> /workspace/sources* directory.

3.5.2 Modifying an Existing Recipe

To modify a new recipe *busybox* to the workspace layer, perform the following procedure:

1. Extract the source files for an existing recipe, execute:

```
$ devtool modify busybox mysources/busybox
```

2. Edit the code and commit your changes to your local git repository.

   You can use any editor to make the changes and save your source code modifications.

3. Build your recipe *busybox* from the workspace, execute:

```
$ devtool build busybox
```

4. Deploy the recipe *busybox* build output to test on the live target machine, execute:

```
$ devtool deploy-target busybox root@<ip of board>
```

   Note: Use command `devtool undeploy-target busybox root@IP` to undeploy and edit the recipe source file again.

5. Apply the changes from the external source tree to a recipe and creates a patch for the committed changes, execute:

```
$ devtool update-recipe busybox
```

   The above devtool command allows the changes to be exported as patches and adds to the recipe. For more information on patching the source for a recipe, see Patch the source for a recipe.

6. Use the devtool reset command to remove a recipe and its configuration from the workspace layer.

```
$ devtool reset busybox
```

3.5.3 Upgrading an Existing Recipe

The devtool upgrade command upgrades an existing recipe to that of a more up-to -date version found upstream. You can use the devtool upgrade workflow to make sure the recipes you are using for builds are up-to-date with their upstream counterparts.

To upgrade a new recipe *busybox* to the workspace layer, perform the following procedure:

1. Upgrade an existing recipe to a new upstream version, execute:

```
$ devtool upgrade busybox
```

   Note: Execute `devtool upgrade busybox --version <version to upgrade> --no-patch` command to upgrade the recipe to the upstream version without applying patches from the recipe to the new source code.

2. Push the source code changes or write as patches on top of the recipe, execute:

```
$ devtool update-recipe busybox
```

3. Build your recipe *busybox* from the workspace, execute:

```
$ devtool build busybox
```

4. Deploy the recipe *busybox* build output to test on the live target machine, execute:

```
$ devtool deploy-target busybox root@<ip of board>
```

Note: Use command `devtool undeploy-target busybox root@IP` to undeploy and edit the recipe source file again.

5. Check the upgrade status of the recipe *busybox*, execute:

```
$ devtool check-upgrade-status -h
```

3.6 Bug Handling Process

3.6.1 Overview

Oniro Project is aiming to build a secure system from the foundation, applying the best industry practices in terms of development quality. However, as in every software projects, bugs do happen. This process explains how we handle bugs.

3.6.2 How to Report a Bug?

If you think you have found a bug in our distribution, please file a bug report in our bug tracker and in the project that you think is the source of the issue. Use the provided template:

- The module affected
- What is the action to reproduce the bug? (Steps to reproduce)
- What is the result you see? (Actual result)
- What is the result you expect? (Expected behaviour)
- Frequency? (always, sometimes, one-time issue)
- Tested version (image name and version, platform)
- Do you know any workaround of this issue? (link to workaround/mitigation steps etc)
- Do you have a fix for this issue?

Developers review the reported issues and perform triage (see below). When a fix is available, the ticket is updated with the details of the solution.

3.6.3 Which Modules do We Support?

We do support all layers included in our reference images and blueprints. It means we accept bug reports in those layers. If the issue affects the upstream part of the layer, we are going to redirect the report to the upstream project and work with upstream on a solution.

3.6.4 Bug Triage

The bug triage is a process where developers asses the bug and set its severity and domain. At the end of this process the bug will:

- Be classified as a security issue, normal bug, feature request, or be rejected if the feature is working as planned or could not be reproduced.
- Have its severity set. Please refer to the documentation of severity levels below.
- Have its domain set. The domains include categories like: toolchain, kernel, Over-the-Air Update (OTA); they can change over time. The bug tracker will include the latest list.

If the bug is classified as a security vulnerability, the engineer assesing the issue will create a new ticket in the private security bug tracker and the discussion will continue in the security bug tracker from that point. Please refer to the CVE Process for details.

If the bug is confirmed as a bug, the developer will assign bug severity: critical, normal, minor or low.

Note: *Critical* severity bugs make a feature unusable, cause a major data loss or hardware breakage. There is no workaround, or a complex one. *Normal* severity bugs make a feature hard to use, but there is a workaround (including another feature to use instead of the desired one). *Minor* severity bugs cause a loss of non-critical feature (like missing or incorrect logging). *Low* severity bugs cause minor inconveniences (like a typo in the user interface or in the documentation).

The bug can originate in a package developed by the project, or from one we use from an upstream source. The process of handling a bug report will change between those two cases:

When the Issue is in the Code Developed by the Project

In the case where the bug originates in the code directly maintained by the Project, the bug is handled directly in the bug tracker.

When the Issue Originates from Upstream Code

If the issue was identified in upstream code, we report an upstream issue in a way appropriate to the upstream project. We store the reference to the upstream bug report in our bug tracker. Depending on the bug severity, we might decide to develop and maintain a fix locally. However, we strongly prefer to upstream the fix first, and then get it with a regular upstream code update.

Please note also that we periodically update maintained packages from upstream sources, regardless of the bugs filled in our system. Our goal is to update to the latest stable version of the package.

3.6.5 Detailed Workflow

Bug Sources

Bugs might be reported by different sources, including Project's own findings (like QA), partner findings, community, or security researchers. There might be also different ways the Project team learns about the issue, including Mattermost channels, discussion forums etc. Issues coming from different sources are centralized in the bug tracker, which also provides an unified identification of all issues.

Acknowledgement and Bug Triage

After the bug is entered, a developer will perform triage. The process starts from acknowledging the issue and then consists of verifying all the information provided by the bug reporter to reproduce the issue. The developer performing triage might ask additional questions. Then they assign severity and domain to the issue in the bug tracker. They also check which versions are affected and might modify the severity level set by the reporter. Any project member, or the bug reporter, who disagrees with the assignment might comment on the issue.

If there is a fix available from the reporter, the developer also verifies if the fix is correct and matches the IP policy. If the fix is judged acceptable, the process might skip to the Releasing step.

We aim at the first answer of the triage (either finishing triage, or additional questions to the reporter) in three working days for critical bugs and seven days for other bugs. In case of a critical bug, the person performing triage informs the maintainers of the affected subsystem.

Prioritizing and Fixing

Bugs with the severity attached enter the prioritization process. It includes a weekly meeting when the team reviews bugs entered or modified during the last week: those during the process of triage, and those with the triage finished. For the bugs with triage finished, the team sets the priority and might assign it to a developer.

The bug fixes should follow the same contributions guidelines as any other contribution. The best practice is to develop a fix for the bug in a separate branch. Fixes for related bugs are possible in the same branch.

Releasing

When a bug fix is available in a branch, the developer creates a merge request. When the change is accepted, it is merged in the main branch. The developer in charge of the bug verifies with the release manager to which branches the change should be backported.

If the bug comes from an upstream project, developers upstream the bug fix. If the upstream is delayed, the Project might ship a local fix. However, we aim at upstreaming all fixes.

During the time of development of the patch and eventual upstream, the developer updates the documentation (if appropriate), and adds a notification to the release notes. Our release notes contain: links to bugs fixed in the release, links to CVEs fixed in the release (publicly known) and a list of CVEs fixed that are still under embargo.

If the bug is classified as critical, it might be decided to perform a separate bugfix release to fix the issue. Otherwise, the bug fix lands in the next bugfix release.

CHAPTER

FOUR

CONTINUOUS INTEGRATION

Oniro Project `git` repositories hosted on https://booting.oniroproject.org/distro use GitLab pipelines for building and testing changes before they are merged. Individual pipelines are documented in each repository, but some general or more important elements are described below.

4.1 The oniro Repository

The oniro repository contains meta-layers specific to Oniro Project and is the most important repository.

The CI pipeline is defined in the file `.gitlab-ci.yml`.

4.1.1 Shared Job Definitions

The *oniro* repository maintains the list of configurations to build. That list includes all the builds jobs, covering all the supported configurations.

The pipeline customizes the `.build` job to allow the build process to take into account any changes being introduced to the *oniro* repository by the incoming pull request. This is done by setting the `CI_ONIRO_GIT_REPO_PATH` variable, which is used by the `.workspace` job defined in the *oniro* repository.

4.1.2 Special Jobs

build-docs

This job runs whenever a merge request is made, or when changes land in the default branch and in addition, the `docs/` directory is modified. This job builds the documentation with sphinx and ensures it can be built without any warnings or errors.

update-docs

This job runs whenever changes land on the default branch and affect either the pipeline or the `docs/` directory. This job *triggers* the pipeline of the `distro/docs` repository, ensuring that any change to documentation present in *oniro* is reflected in the aggregated documentation build maintained in the *docs* repository.

4.2 The docs Repository

The docs repository contains the general documentation of the Oniro Project project. The documentation is normally written in reStructuredText, with an important difference. Unlike in a regular project, the documentation here is not standalone. Instead, the git repository contains symbolic links that are valid when an entire workspace is constructed with oniro repository.

This complicates the testing process and the deployment process, but allows the resulting documentation to span multiple repositories, permitting code and text to be conveniently changed in one go.

4.2.1 Special Jobs

build

The `build` job ensures that the documentation can be built with *sphinx-build* without any warnings or errors. Apart from the complexity of setting up the workspace as described above, it is fairly typical.

deploy

The `deploy` job builds aggregated documentation view that is rendered at https://docs.oniroproject.org/. This job effectively constructs the workspace as described above and then archives the entire documentation source code, dereferencing any links that were followed to other repositories, and commits the updated set of files to helper repository which is observed by readthedocs.

The helper repository is called oniro-readthedocs-aggregated. That repository contains a webhook, configured at the level of the GitLab project, which asks readthedocs to re-build the documentation.

4.2.2 Implementation Highlights

Commit Credentials

Commit to the aforementioned aggregated repository is allowed by a personal access token that is set up in that repository. The value of the token is stored as a protected variable available to the docs repository.

4.3 On-device Testing

4.3.1 Overview

Oniro Project implements distributed device testing using Linaro Automation and Validation Architecture (LAVA). This architecture creates an environment where you can operate the necessary physical infrastructure responsible for testing development on real devices, like operating system boot-loader and kernel development, while sharing access to a project-specific software infrastructure used in the public cloud.

4.3.2 How does the CI System Work?

The system automatically performs a set of test jobs upon a new or modified merge request. Failed jobs stop the pipeline, allowing you to review build logs, reproduce and resolve the failure locally. The central system maintains a queue and schedules build and test jobs for the available workers. The workers may be auto-scaled, for example, virtual machines in the public cloud, or fixed, for example, a set of physical machines prepared for automated deployment and testing.

4.3.3 Testing Infrastructure

The testing infrastructure consists of a pool of devices physically located at a specific site. These devices are operated and maintained by partner companies and/or individuals. The device maintenance may involve resolving networking problems, swapping out a faulty storage medium, or configuring the device for initial provisioning to the pool.

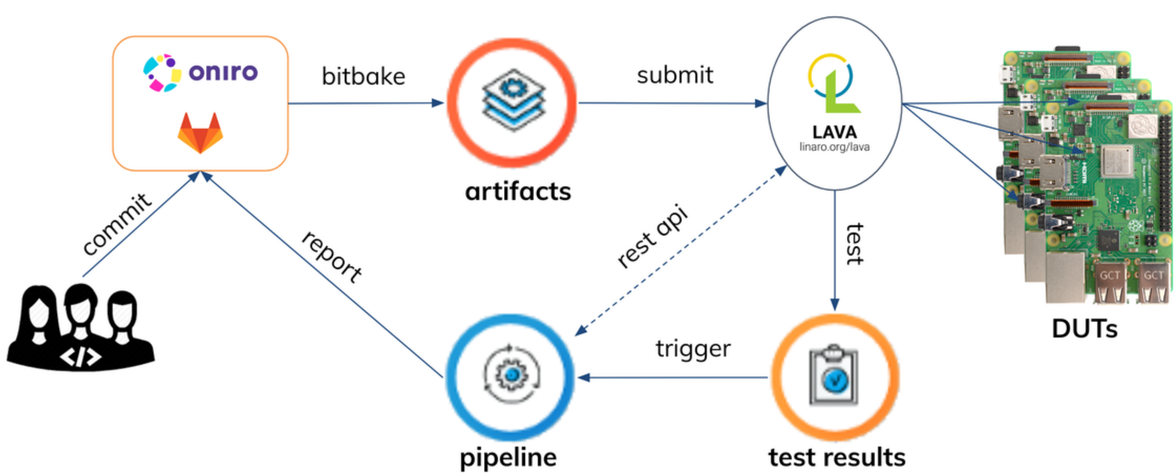
A site may operate as little as one device or as many as several dozen or hundred, depending on the test suites load and available resources. Sites can thus range from a single desk with a single device, a small rack with several devices in a corporate office, up to a dedicated testing lab with a large number of diverse devices.

Each site is added to the central infrastructure by registering a software-specific service operating on-site and connecting it to the central system. There are two possible site configurations, depending on connectivity to the public cloud:

- If the network connection is poor, building and downloading the images locally is suggested.
- If the network connection is robust, use the central build system directly to build the images, where scalability is easy.

Typically a micro-site that has limited throughput will be bound by the limited number of test devices and will be able to perform the builds locally much faster than being able to pull each new large image from the central system. Typically micro-sites will also see a more limited usage, for example, to support a bootstrap of a new project or preparing the process for automation for a new device.

4.3.4 Oniro CI Flow



Oniro CI Flow

1. New merge request triggers the CI pipeline.
2. The pipeline build stage builds and uploads artifacts.
3. Test job under test stage is triggered once the dependent build job finishes.

4. The LAVA job definition template populates the required variables for executing the test job using the values from the GitLab CI.
5. The test job is submitted using the REST API, and the LAVA executes the job.
6. Report job is added to the pipeline, if all the dependent test jobs are added.
7. After all LAVA test jobs for the pipeline finish, using the callback system, LAVA triggers the last CI job in the loop, the report job.
8. The report job calls LAVA REST API to collect the test results.
9. The report is submitted back to GitLab, and the test report can be seen in the merge request.

4.3.5 Setting up Remote LAVA Worker

A LAVA worker performs test execution in LAVA. LAVA workers can connect to the LAVA server on a local network or using the internet. We called it `remote worker` as it can be located anywhere that has internet access.

A remote LAVA worker can be deployed manually by following this guide Running the LAVA dispatcher in a Docker container. The process has been automated using ansible as well.

Following are the steps for automatic deployment.

1. Install Debian buster on your worker.
2. Clone LAVA Ansible Playbooks repository.

```
git clone https://git.ostc-eu.org/OSTC/infrastructure/lava/lava-playbooks.git
```

3. Define a group of worker/workers in `hosts` and related group variables under `group_vars`.

   `hosts` example:

```
[worker_dev]
# local VMs for dev
192.168.56.103 worker_name=<hostname.lab_name> server_token=<worker_token>
192.168.56.104 worker_name=<hostname.lab_name> server_token=<worker_token>
```

   Worker token is generated on LAVA server. Ask LAVA administrator to create your worker and share authorization token.

   `group_vars/worker_dev.yaml` example:

```
lava_config_repo: https://git.ostc-eu.org/OSTC/infrastructure/lava/lava-playbooks.
↪git
lava_config_repo_branch: master
# sudo user without password
ansible_user: <username>
server_url: http://192.168.56.102
```

4. Run Ansible playbook for worker deployment.

   Deploy to a worker group:

```
ansible-playbook -l worker_dev playbooks/deploy-docker-worker.yaml -b
```

   Deploy to a specific worker:

```
ansible-playbook -l 192.168.56.103  playbooks/deploy-docker-worker.yaml -b
```

5. Check service status with command `systemctl status lava-docker-worker.service`.
6. You can also optionally provide arbitrary Dockerfile to LAVA docker worker.

4.3.6 Adding a Device to LAVA

Test devices are connected to their local LAVA worker and manged by LAVA server. The following wiki pages have been created for adding devices to LAVA.

- How to deploy wic image on Raspberry Pi in LAVA
- Adding Arduino Nano 33 BLE Board to LAVA Lab
- Adding Nitrogen Board to LAVA Lab
- Adding Avenger96 Board to LAVA Lab

4.3.7 References

- For more details on LAVA, see Introduction to LAVA.
- For configuring and adding a new device, see Adding your first devices.
- For more details on LAVA test job definition template, see lava-test.
- For sample test job definition, see sample jobs.
- For our CI test job definition, see CI jobs.
- For LAVA supported device list, see Supported Devices.

CHAPTER

FIVE

BUILDING ONIRO PROJECT DOCUMENTATION

This topic outlines the standard way of generating the Oniro Project project documentation locally using the source files available in the booting.oniroproject.org repository which aggregates documentation from multiple other components.

5.1 Overview

The Oniro Project documentation is written in reStructuredText markup language (`.rst` file extension) with Sphinx extensions to generate a structured stand-alone website.

5.2 Prerequisites

To generate the HTML documentation locally, you need to have the following packages pre-installed in your system:

- Sphinx (for more details on Sphinx installation, check Sphinx Getting Started documentation)
- The following Sphinx Extensions
  - sphinx-tabs (supports tabbed content in the documentation)
  - sphinxcontrib.plantuml (supports plantuml content)
- Plantuml (supports UML diagrams)

  The method of installing the *plantuml* package is dependent on your Linux distribution. For example, on a Ubuntu host, you can install *plantuml* using the official package repository:

  ```
  $ sudo apt-get update -y
  $ sudo apt-get install -y plantuml
  ```

- Make (to build the documentation using the provided Makefile)

5.3 Building the Documentation

To generate a local copy of Oniro Project documentation, perform the following steps:

1. Create a local workspace and clone the Oniro Project project files to your local, refer to *setting up a repo workspace* section for more information.
2. To generate output as HTML, run the following command:

```
$ make
```

The HTML output is built and can be viewed in your browser from the *<docs repository>/build/index.html* path.

Note:

- All the local Sphinx warnings and errors generated during the build process must be fixed and validated.
- To validate the changes, execute `make clean && make` command to generate a clean HTML output.

5.4 Reference

https://www.sphinx-doc.org/en/master/contents.html

https://www.sphinx-doc.org/en/master/usage/restructuredtext/index.html

CHAPTER

SIX

INTELLECTUAL PROPERTY COMPLIANCE POLICY

6.1 Open Source Policy

6.1.1 Introduction: Why Open Source Values are Our Values

In a **connected world** like the one we live in, we believe that the persistence of business models based on closed source software, silo software development, lack of interoperability and vendor lock-ins is causing substantial damages to users, content providers and device manufacturers in terms of transaction costs and consequently lost opportunities.

The incredible potential offered by the digital revolution is being stifled by the incapability of smart devices from different brands to interact seamlessly and to ensure interoperability; requiring strong efforts sometimes unreasonably in making applications and contents compatible with different platforms and devices – while the same energy could be better used to improve them, and make users' lives easier.

Today's enhanced device and ecosystem diversity does not ensure freedom of choice: users are not free to combine different devices with each other; device manufacturers are not free to choose software components based simply on their quality and features; content creators are not free to distribute the same content or application to any device and platform.

In opposition to this model, we are building **Oniro Project**, a **fair and open source software ecosystem**

We truly believe that such an ambitious project can succeed only if it is true open source.

"True open source" does not mean identifying the problem, building the solution and donating that entire solution to the world. Rather, "true open source" is about **collaboration**, it is about **sharing** and discussing **ideas, plans and roadmaps** with others.

"True open source" is about **active and responsible members of a community** who share some **common fundamental values**: the freedom to use, study, share and improve software programs; the freedom of choosing technologies based only on their features and quality, and not because of vendor lock-in strategies; the value of interoperability, as a means to achieve such freedom; the values of shared learning, peer review and meritocracy, as a means to enhance developers' skills – and get better software, too; the value of reusing others' code while respecting their rights, in order to build a true software commons; the value of transparency, to share control on technology and protect everyone's digital sovereignty.

This policy is about how we want to implement these values in our organization and in the software projects we steward.

6.1.2 Scope

Objective Scope

This policy covers **Oniro Project** and any other open source project related to Oniro Project ecosystem, hosted and maintained by Oniro Project Working Group (**Oniro Project Working Group**).

Subjective Scope

This policy is binding for Oniro Project Working Group's developers, collaborators and other individuals or entities connected to the development of software in or for Oniro Project Working Group. External contributors and community members shall undertake and fully commit to follow its guidelines and principles (and particularly those set out in sec. *4.2* and in sec. *5.2*) when they are collaborating to projects related to Oniro Project.

Entry Into Force

This policy is first published on January 15th 2021.

Implementation of this policy within Oniro Project Working Group's organization will require a preliminary stage involving the allocation of internal resources with the appointment of the required roles, the adjustment of internal procedures and workflows, the set-up and testing of compliance toolchains, the training of the affected staff members.

After completion of such preliminary stage, this policy will definitively come into force on March 22nd 2021.

6.1.3 Glossary

Inbound, Outbound license Respectively, the license used by an upstream project whose software is included in the developed combination, and the license which is used by the project when distributing the software;

Inbound (in)compatibile (license) License of A is inbound incompatibile with license of B when license of B does not tolerate including A in an A+B combination licensed under the license of B.

Outbound (in)compatibile (license) License of A is outbound incompatibile with license of B when license of A does not tolerate A being included in an A+B combination licensed under the license of B

Derivative Derivative is the same concept and shall have the same meaning as in software copyright. For the technical aspects, we mainly refer to the official guidelines and statements of the FSF (particularly, the statements about the '(mere) aggregation' concept that can be found both in the FAQ on GPLv2 and in the FAQ on GPLv3).

Mere aggregation A combination of two or more software artifacts (eg. applications, libraries, kernel, etc.) in the same distribution without one being the derivative of the other, despite – in case – one cannot work without the other, but they operate at arm's length (e.g., through interfaces, sockets, filesystem, database tables).

Own software Software whose copyright is owned by the entity distributing or using the same software.

Upstream, downstream Direction from and to which software and other information or technology flows in a chain of interaction from the originating artifacts and their modifications until they reach the final distribution outlet. If entity A provides technology to entity B, that receives and transforms it, A i upstream to B and B is downstream to A.

Compliance artifacts [from Openchain 2.0 defintion] a collection of artifacts that represent the output of the OpenChain-compliant process for the published software. The collection may include (but is not limited to) one or more of the following: source code, attribution notices, copyright notices, copy of licenses, modification notifications, written offers, Open Source component bill of materials, and SPDX documents.

Open Source License In descending order of preference *(a)* an OSI-approved license, or *(b)* a license qualifying as free software according to the list published by the Free Software Foundation (FSF), or *(c)* a license falling within the Open Source Definition and/or within the FSF definition of Free Software, **but** which is neither OSI-approved nor expressly listed by FSF (note that this latter option requires a previous assessment by the internal Legal Team to be considered "open source")

License Proliferation Tendency to create an unjustified high number of licenses, often for one single project, causing increased friction, uncertainty, incompatibilities, for no good reason.

6.1.4 Upstream First...

Oniro Project software projects typically incorporate code from other open source projects (such as software libraries, kernels, system tools, etc.) on the upstream side of the supply chain, while on the downstream side they are intended to be used as a basis to develop vertical-specific implementations (like firmware for specific IoT or smart devices).

We at Oniro Project Working Group choose to adopt an "**Upstream First**" **approach**, both for our own projects and for third party projects we incorporate in Oniro Project.

...for our own code/projects

For projects we directly maintain, "*Upstream First*" means that even if there are any downstream versions developed by Oniro Project Working Group (vertical-specific, device-specific, etc.), any improvements, bug/security fixes and new features will be generally made available first on the corresponding upstream projects.

Exceptions to this general principle may be allowed in case of changes required to fix embargoed and/or critical CVEs, or involving customer proprietary information or experimental features, or in case of product deadlines requiring to work on upstream and downstream at the same time.

For this reason Oniro Project Working Group developers should verify with their managers if any of the above exceptions apply before pushing any changes upstream for the first time.

...for third party projects we incorporate in our projects

For third party OSS projects we incorporate or include in our projects or software distributions, the general principle is to avoid downstream forks, if at all possible: any improvements, bug/security fixes, new features should be generally offered first to the upstream project, requiring to be accepted.

Only if changes are not accepted upstream within a reasonable time, Oniro Project Working Group will take actions without delay in order to work out a viable option, taking into account long-term maintenance costs of the fork, possible loss of interoperability, and any other relevant technical reasons.

Furthermore, the same exceptions listed in sec. *4.1* above apply here, and may justify downstream forks of third party projects.

In any case, **downstream forks of third party OSS projects** included in Oniro Project should follow common best practices, so that:

- upstream original code and downstream modifications are clearly identifiable from each other (eg. by means of patch files and/or adequate git branching policies);
- the authors of downstream modifications are clearly identified (this includes also original upstream authors in case of backports from upstream project's next versions);
- downstream version tags should be coherent with upstream version tags and should follow semantic versioning specifications and best practices, preferably using build meta tags according to semver spec item-10

For instance, if the upstream project 'foo' has a version tag like '1.0.0', the corresponding forked version of 'foo' should have a tag like '1.0.0+|main_project_tag|.1')

6.1.5 Collaborating

Open Source is about Collaboration

Open source, it goes without saying, is about collaboration. Oniro Project Working Group is committed to being available as much as possible and practicable in making their open source software truly accessible and easily reusable by anybody in the downstream part of the supply chain.

At the same time, open source collaboration is unique in that the licensing *per se* is sufficient for a project to be enabled to *receive* contributions from the downstream, and any entity in the ecosystem can be both upstream and downstream at the same time without the need to have any interaction with the project other than at commit time. The only necessary interface to give and receive software contributions both upstream and downstream is the license.

OpenChain facilitates the interaction between organizations. It does so by internally guiding Oniro Project Working Group deploying workable rules, principles, knowledge. it also provides evidence and interfaces to interact with downstream and upstream entities for any activity involving the software. The Oniro Project Working Group has undertaken a path to **conform to OpenChain 2.0 specifications (ISO/IEC 5230)**, which is planned to be completed by the end of 2021.

Some projects, especially corporation-controlled or sponsored, have developed policies requiring that all the software be centralized with only one copyright holder, either by appointing it the full owner or just a fiduciary owner on behalf of the community. This generally requires signing a legal deed, often referred to as CLA (short for "Contribution Licensing Agreement") or "CAA" ("Contribution *Assignment* Agreement"). Other projects prefer a more lightweight approach, and only require a general reassurance that the contributor is identified and accepts the terms of licensing (it is the case of DCOs).

How others can collaborate in our projects

DCO

All contributions must be signed off by the contributor in each commit with a `signed-off-by:` and the name of the author of the commit. Oniro Project Working Group only accepts **pseudonyms** in case the author of the commit is well-known under that name. In case of doubt, the authors must use their real name and an email address that they control.

Signing off the commit means that the author commits and declares what is stated in the Linux Foundation's Developer's Certificate of Origin (or "DCO"). Each repository must have a `contributing.md` file that reflects this policy. All contributions that do not contain a sign-off statement will be rejected or will receive a request to sign-off.

Meritocracy

All contributions to projects that Oniro Project Working Group stewards – or in which it has directional power – follow a principle of meritocracy and non discrimination. This means that contributions will, as much as possible, be dealt with as equal, irrespective of who has contributed them and decisions or votes shall be taken in the most dispassionate way. Oniro Project Working Group places a high value and consideration on the need to be impartial when deciding to be involved in a third party's project, and therefore adheres to the same principles itself.

Oniro Project Working Group considers the Open Decision Framework a good reference for the above process. Other guidance documents may be adopted on a case-by-case basis to better fit in the organizational and cultural environment where Oniro Project Working Group operates.

6.1.6 Creating New (Sub)Projects

Reuse first: is there really the need for a new project?

When developing new components and functionalities, before deciding to start a new project developers are encouraged to first consider existing open source alternatives, following the criteria set out in sec. *7* .

License choice

If no viable existing solution is found, the project team will have to clarify from the very beginning the applicable license for the new project which should be approved by the internal Legal Team and the internal IP experts.

Any Oniro Project-related project should be open source by default.

As a general rule, Oniro Project-related projects should default to "Apache Public License v. 2.0", unless there is a proven and justified reason, discussed with the internal Legal Team. There may be cases in which we must take into account licensing preferences and expectations of a project's target contributor and user community, and possible legal constraints (eg. as a general rule, Linux kernel modules should be licensed under GPLv2 because that is the license of the Linux Kernel and it is a strong copyleft license).

In any case, the chosen license must be an Open Source License according to the definition given in *Glossary* , taking into consideration the guiding principle so as to avoid as much as possible license proliferation as defined in *Glossary* , by using only well-known open source licenses, and by avoiding creating new open source licenses or new license exceptions at all.

Codes of Conduct

We believe that any contributor (internal or external) to Oniro Project projects should have the right and duty to interact in ways that contribute to an open, welcoming, diverse, inclusive, and healthy community.

As a means to achieve this, including a code of conduct along with a new project's license is mandatory.

As a general indication, we suggest Oniro Project Working Group developers to use the Contributor Covenant 2.0).

6.1.7 Incorporating Third Party Projects

When planning to add new components and functionalities, Oniro Project Working Group developers are strongly encouraged to first consider existing open source alternatives.

However, not any open source software is acceptable for inclusion in projects maintained by Oniro Project Working Group.

- *Zero*, the component must be open source, i.e. it must come with an open source LICENSE file (see next point), and the component should not contain any known critical CVEs (see last point).
- *First*, the license: it should be an Open Source license, according to the definition given in *Glossary* . Should the considered project contain components subject to different licenses, their internal coherence should be checked, to avoid inbound/outbound license incompatibilities (see *Glossary*).
- *Second*, dependencies and license compatibility: if the considered component will be a dependency of, and/or will depend on other components, inbound/outbound license compatibility should be checked (see *Glossary*). Also other possible legal issues should be considered (for instance related to possible third party patents covering the technology).
- *Third*, it should be checked if the project is maintained by an independent community, or by a single corporation or a foundation, weighing the pros and cons of it (eg. financial and organizational support vs. possible lack of independence from private interests that may diverge from Oniro Project goals).

- *Fourth*, project maturity and activity: long established projects, with frequent substantive commits, should be generally preferred over projects that are relatively new or have been inactive for a long time.
- *Last but not least*, quality and security: projects with a relatively high number of critical open bugs, issues and CVEs should be generally avoided.

6.1.8 Open Source License Compliance…

…is a Matter of Respect

As mentioned in the *Introduction* , "True Open Source" means respecting third parties' rights while reusing their code – which, in legal terms, means respecting open source license obligations – whilst acknowledging their efforts and respecting the ideals of the Open Source community.

As explained in sec. *5.1* , in an Open Source ecosystem **the license is the key legal interface enabling parties to give and receive software contributions**, upstream and downstream, from entities and individuals spread across different countries and organizations. **Without the license, there is no** "**True Open Source**", both from a legal and a community perspective.

Thus, respecting the license essentially means respecting the legal terms that enable collaboration in an open source community; if one wants to engage or participate in a community, one should respect its rules, not just because they are legally binding, but to gain trust and respect, and be fully accepted into that community.

OSS compliance in our upstream projects…

Our "Own" Code

As to our own open source code included in Oniro Project projects and software distributions (for a definition of "own" software, see the *Glossary* , we commit to always make available the complete source code in our public repositories, following the "*Upstream First*" approach (see sec. *4*).

We also commit to be fully REUSE Compliant, in order to cut off any uncertainty about copyrights and licenses applying to our own code, therefore easing open source compliance work for Oniro Project downstream implementers.

Third Party Code

Given the size and nature of Oniro Project project, a number of these components typically come from third parties.

From a copyright point of view, some components may be independent works in themselves – or, in the GPL parlance, "mere-aggregation" artifacts – while some others may form a derivative work, depending on the technical aspects of their dependency relationship (see *Glossary*). In the case of derivative works, not all combinations of components are allowed: inbound component licenses must be compatible with each other *and* with the outbound license of the whole derivative work (see *Glossary*), and, in case of license incompatibilities, the whole derivative work cannot be distributed (because it would violate the license terms of one or more components).

Therefore, with regards third party open source components included in Oniro Project projects, we commit to **perform a dependency mapping and a legal analysis on third party code on a regular basis via CI/CD**, in order to **avoid license incompatibility problems from the earliest possible moment during project development** and resolve eventual incompatibilities if they are seen to arise. Such legal analysis will also enable us to **identify and meet** (and to help downstream implementers to meet) all **license obligations related to third party components**.

Accepted Licenses

As a general rule, while we give preference to "Apache Public License v. 2.0", we will accept any licenses for third party components to be included in Oniro Project projects and software distributions, as long as they are Open Source Licenses, according to the definition given in *Glossary* .

Instead of prohibiting particular licenses to be included in Oniro Project, we commit to provide our developers with guidelines and procedures to avoid prohibited *combinations* of software components due to license incompatibilities, as described above.

OSS compliance for Downstream Implementers

We commit to provide device manufacturers and other Oniro Project downstream implementers with metadata and open source tools to ease their compliance work and to help them generate BOMs and compliance artifacts (see *Glossary*) for their software and firmware distributions.

We are aware that some downstream implementers may need, for legal or other internal reasons, to rule out components subject to certain open source licenses from their software/firmware distributions. Thus we commit to include in Oniro Project appropriate tools and configuration options to that purpose, and to make reasonable efforts to provide if possible a choice of alternative components subject to different licenses, whenever it is reasonable and technically feasible to do so.

Reproducible Builds

Oniro Project Working Group commits – whenever it is reasonable to do so – to follow the best practices provided by the Reproducible Builds project.

6.1.9 Proprietary Drivers and object code

The rule

Oniro Project strives to be 100% Free and Open Source Software.

However, in certain cases, there is no viable alternative than accepting a **proprietary, binary-only** contribution – at least as a temporary solution, such as proprietary firmware. This should be an exception and it needs to be deeply reviewed by the Open Source Executive Committee.

Generally, no proprietary software may be included in Oniro Project, and in any case no proprietary software that may lead to license incompatibilities (eg. kernel modules).

When *both* proprietary and FLOSS drivers are available for a specific device, upstream Oniro Project distribution will include *only* FLOSS drivers (even if they are less performant or have less functionalities); instructions to download proprietary drivers downstream may be provided to device makers, but

- they must be downloaded separately by the downstream implementer
- the proprietary driver must be known not to raise compliance issues for those downloading it: it shall however be the downstream implementer's duty to check thoroughly and a full disclaimer must be included that **due diligence** is upon them
- Oniro Project never knowingly advises to use malicious software or software containing know major vulnerabilities
- the rest of the policies with regard to third parties software must be complied to the maximum extent

- the downloadable software must not be a dependency of Oniro Project under no conditions: if this is the case, sec. *9.2* should be followed.

The exception

Exceptionally, proprietary firmware/bootloaders may be included in Oniro Project for hardware compatibility reasons only (i.e. when there is no other viable option to get a build that manages to boot on a specific device), in one of the following ways, in this order of preference:

a) Convince the IP vendor to upstream the firmware to some package such as `linux-firmware` that allows for redistribution;

b) Enter into licensing agreement with the IP owner in order to acquire sufficient redistribution rights and the right to pass them through; this must be sufficient so that Oniro Project images can incorporate the proprietary firmware and boot out of the box.

c) **If nothing else is workable**, a scripted workflow that allows a user to download Oniro Project in a temporary version, which may not even boot on a device, download the proprietary firmware on the user's machine, prompt the user for an EULA, and then incorporate the firmware into the image. The last part is usually achieved by loop mounting the image locally and adding the necessary files. The presence of this download mechanism and the requirement to sign an EULA **must be clearly advertised at download time**, at latest.

Notices

The presence of proprietary files must be sufficiently advertised in the repository or in the repository's directory where firmware is placed.

6.1.10 Patents

General

In several jurisdictions, software is also covered by patents. Therefore a license from the patent(s) holder is necessary for a downstream implementer.

This is in its essence contrary to the spirit open source licensing and a full release of software as open source should be made in a way that patents cannot be used to revoke the liberties that the license under which the software is released have granted.

In this section we discuss the ways patents play a role in Oniro Project Working Group's distribution of open source software.

Open Invention Network

OIN pools patents that are relevant to the specific areas covered by the Linux Definition, where the patent holders agree on a non aggression pact with all other members, creating a patent-war-free-zone limited to the scope of the Linux Definition. Oniro Project Working Group endorses OIN and invites all players in the Open Source field to join the project.

Open source licenses are (also) patent licenses

Often, open source licenses are referred to as "copyright licenses". While extensively using copyright as a tool to grant and protect the freedoms embedded in the Open Source and Free Software Liberties, open source licenses are to be seen as conditional permissions to use the software. No matter under whatever exclusive right this permission is required, insofar as it is controlled by the licensor.

In this light, the main license chosen by Oniro Project Working Group is the "Apache Public License v. 2.0" license. This license embeds an express patent license over the patents held by the contributor for the contributions they made.

Other licenses include some sort of patent pledge, permission, grant, license.

6.1.11 Trademarks

Ownership of signs

Despite the code of Oniro Project being Free/Libre and Open Source Software, it is important that the trademarks and the other signs be preserved in their function to identify this project. Therefore, although everyone has the right to benefit from the liberties as provided by the licenses on the code and upon compliance with their conditions, at the same time exercising these liberties does not imply nor require the use of trademarks.

Publication of the source code of Oniro Project and of any other elements (there included images, logos in vectorial and raster format, etc.) does not imply a license to the use of trademarks and of any other identifying sign, including the specific combination of colors ("trade dress"), be they registered or unregistered signs.

The only permitted uses are those explicitly provided for and are expressly subject to the relevant trademark policy or license, if offered.

Permitted uses

Mirroring and forks

A use of the distinctive signs in a reasonably updated copy of the official repository of Oniro Project is always permitted, conditional upon:

- a clear mention be made and an hyperlink be offered in a sufficiently prominent way to lead to the official repository, mentioning at the same time, for example in an added "mirror.txt" file, that it is a copy of the official repository, offered for convenience, or a fork.
- if it is a fork of Oniro Project to host future independent developments to be conferred into Oniro Project, the `master` branch be at all times in alignment with the official repository and all contributions be merged to a non-owned branch only through a pull request to the official repository, and never through a local merge and push.
- for the sake of clarity, nothing in these rules has either the scope or the effect to limit or prevent anybody from the full exploitation of the freedoms and rights conferred by the applicable license to any portion of Oniro Project

Descriptive use

In general, according to the relevant law (please check) it is permitted to use signs only in a descriptive way, that is to mention Oniro Project or certain elements thereof, under the condition that there is no arising potential confusion as to the provenance of the code, that there is no interference with the normal exploitation of the signs in their distinctive function, there is no dilution of them and – as a general rule – good faith is used. As a general rule, it is not considered in good faith to use colored logos, using the names in an identifying way without the use of clarifying elements of the descriptive use.

This section has only explanatory purposes of what the law generally considers fair use and how Oniro Project Working Group interprets it, but it bears no effect whatsoever to acknowledge any fair use beyond what the law already considers so and cannot be otherwise construed.

Software heritage

All uses connected to preservation of software as cultural heritage, in accordance with the Paris Call are always permitted.

Third parties' TM

This policy document has no effect over trademarks of any nature belonging to third parties.

6.1.12 Governance

Oniro Project Working Group commits to design and implement good governance processes to ensure that the principles and values expressed in this policy are followed over time within our organization.

All governance processes will be designed having in mind the principles set out in sec. *5.3* . It is understood that OSS compliance is a continuous process, that in the context of Oniro Project needs to be run in parallel with the development process.

To achieve this, we commit to avoid over-engineering of governance processes, and to make use of ticketing systems, CI/CD facilities and DevOps-style workflows as much as possible, whilst giving developers access to internal legal and other OSS-related organizational resources.

Oniro Project Working Group developers and decision-makers shall adhere to OSS Policy Implementation Guidelines that are internally published from time to time, and use software tools and technical facilities provided by those guidelines when working on projects related to Oniro Project.

To this purpose, Oniro Project Working Group developers and decision-makers will be provided with continuous training programs on the content of this policy and the related implementation guidelines.

6.1.13 Copyright Notice.

Authors: Carlo Piana and Alberto Pianon (Array)

Reviewers: Davide Ricci and Christian Paterson (Huawei)

6.2 Open Source Policy Implementation Guidelines

6.2.1 Introduction and Scope

Open source compliance is the process by which users, integrators, and developers of open source software – when reusing third parties' open source software for developing and distributing their own software – ensure to meet and comply with the license conditions imposed by the original copyright holders.

In the context of the Oniro Project project, main objectives for open source software (OSS) compliance for Oniro Project Working Group are:

- comply with open source licenses of third party components included in Oniro Project;
- license Oniro Project Working Group's copyrights on Oniro Project as open source in a correct and unambiguous way;
- protect Oniro Project Working Group's, Oniro Project Working Group customers' and third parties' proprietary IP not intended to be released as open source;
- facilitate the effective use of Oniro Project by downstream implementers / device makers by providing them with reliable license and copyright metadata and with tools to produce Compliance Artifacts for their products.

For the above purposes, Oniro Project Working Group adopted a general Open Source Policy, addressed both to Oniro Project Working Group's internal developers and to external contributors to projects related to Oniro Project ecosystem.

These Open Source Policy Implementation Guidelines, addressed to Oniro Project Working Group's internal developers only, are aimed at capturing all requirements that need to be implemented in order to have a fully functional Open Source compliance Program within Oniro Project Working Group in conformance with the OpenChain Specification 2.0.

As the main Open Source Policy, these implementation guidelines cover Oniro Project and any other open source project hosted and maintained by Oniro Project Working Group (Oniro Project Working Group), and related to Oniro Project ecosystem.

6.2.2 Definitions

> This section is taken from Section 2 of the OpenChain Specification 2.0, Copyright 2016-2019 Linux Foundation, licensed under CC-BY-4.0

Compliance Artifacts a collection of artifacts that represent the output of the Program for the Supplied Software. The collection may include (but is not limited to) one or more of the following: source code, attribution notices, copyright notices, copy of licenses, modification notifications, written offers, Open Source component bill of materials, and SPDX documents.

Identified Licenses a set of Open Source Software licenses identified as a result of following an appropriate method of identifying Open Source components from which the Supplied Software is comprised.

OpenChain Conformant a Program that satisfies all the Requirements of this specification.

Open Source software subject to one or more licenses that meet the Open Source Definition published by the Open Source Initiative OpenSource.org or the Free Software Definition (published by the Free Software Foundation) or similar license.

Program the set of policies, processes and personnel that manage an organization's Open Source license compliance activities.

Software Staff any organization employee or contractor that defines, contributes to or has responsibility for preparing Supplied Software. Depending on the organization, that may include (but is not limited to) software developers, release engineers, quality engineers, product marketing and product management.

SPDX the format standard created by the Linux Foundation's SPDX (Software Package Data Exchange) Working Group for exchanging license and copyright information for a given software package. A description of the SPDX specification can be found at www.spdx.org.

Supplied Software software that an organization distributes to third parties (e.g., other organizations or individuals).

Verification Materials materials that demonstrate that a given requirement is satisfied.

6.2.3 Requirements

Program Foundation

Policy

All joining Software Staff to Oniro Project Working Group will be made aware of the existence of the Open Source Policy and of these Implementation Guidelines, and associated training policy and its location during the induction process. This will be recorded on the Induction Checklist of HR system.

Roles and responsibilities

A general list of useful contacts with corresponding operations and areas related to Oniro Project Working Group can be found in the Appendix to these Guidelines.

A list of roles with corresponding primary responsibilities, main competencies and time requirement for the different participants in the Program can be found in the table below:

Table 1: Primary roles and responsibilities

Role	Primary responsibilities	Main competencies and understanding	Time requirement
Open Source Review Board (OSRB) (core team)	• Ensuring compliance with open-source software licenses • Facilitating effective usage of and contributions to open-source software • Protecting proprietary intellectual property from unintended disclosure • Establish the Compliance End-to-End Process • Create and maintain compliance policies, processes, guidelines, templates, and forms used in the compliance program • Review proposals for the incorporation, modification, and distribution of open-source components • Coordinate software audits	It consists of representatives from legal, engineering, and product teams, in addition to the Compliance Officer	N/A

continues on next page

Table 1 – continued from previous page

Role	Primary responsibilities	Main competencies and understanding	Time requirement
Open Source Executive Committee (OSEC) (extended team)	• Setting open-source strategy • Reviewing and approving the release of intellectual property • Launching new open-source projects • Handing over open-source projects to other entities (foundations etc.)	It consists of engineering, legal, and product marketing executives in addition to the Compliance Officer.	N/A
Compliance Officer (core team)	• Chairing the OSRB and the OSEC • Driving all compliance activities • Coordinating source code scans and audits • Coordinating distribution of source code packages • Contributing to compliance and OS training • Contributing to improving compliance program • Reporting to OSEC on compliance activities	• Cross-functional background on engineering, marketing, and business development • Strong technical / engineering background to engage directly with engineering teams, development partners, and open-source community • Ability to conceive and implement internal and external processes cross-functionally • Primary internal and external evangelist for open-source • Strong existing relationships with relevant open-source communities, industry consortia, and open-source foundations • Solid understanding of common open-source licenses to discuss with Legal Team • Knowledge of industry practices	Full-time, combined with the role of Open Source Director
Engineering / Product Team Representative (core team)	• Participating in OSRB and OSEC • Ensuring that team managers and developers follow compliance policies and processes • Integrating compliance practices in the development process • Contributing to improving the compliance program • Responding quickly to all questions • Conducting design, architecture, and code reviews • Preparing software packages for distribution • Integrating compliance milestones as part of the development process	Internal open-source compliance training	Combined with other roles

continues on next page

Table 1 – continued from previous page

Role	Primary responsibilities	Main competencies and understanding	Time requirement
Legal Team (core team)	• Participating in OSRB and OSEC • Advising on usage, modification, distribution of open-source software • Providing guidance on licensing • Contributing to and approve training • Contributing to improving the OS compliance program • Reviewing the content of OS portals • Reviewing lists of license obligations to fulfill • Drafting and reviewing open-source notices	• Strong legal expertise in IP law in the software field and particularly in the open-source compliance field • Deep understanding of technical issues relevant to open-source compliance (especially as regards software component interaction, derivative works and inbound/outbound license incompatibilities) • Deep understanding of the dynamics of the open-source software sector and its business models, to make company's IP licensing consistent with them	External team, coordinating with internal legal dept
Audit Team (core team support)	• Performing audits on open-source software on a CI/CD basis to identify applicable licenses and copyright owners, and spot potential legal issues to be reported to the Legal Team • Contributing to the development of existing and new tools to facilitate compliance automation	• Cross-functional background in software engineering and open-source licensing • Adequate knowledge and understanding of common open-source licenses • Adequate understanding of software component interaction and software build processes in different programming languages, and of their legal implications of them in terms of derivative works and license compatibility • Expertise in using (and possibly contributing to the development of) common OSS compliance software tools	Full-time

continues on next page

Table 1 – continued from previous page

Role	Primary responsibilities	Main competencies and understanding	Time requirement
Community Manager (extended team)	• Promoting open-source projects and grow their user base • Partner and ecosystem development (developer and contributor community) • Maintaining community health (friction reduction, conflict resolution, code of conduct enforcement, etc.) • Managing conflicts and tensions between community users and developers on the one hand, and commercial device makers and product customers on the other hand • Integrating community goals in product value proposition • Reporting to OSRB and OSEC	Cross-functional skills in marketing, partner management, developer management, product strategy, and product management	Full-time
Documentation Team (extended team)	Including open-source license information and notices in the product documentation	Existing roles in charge of maintaining product documentation	As part of their role
Information Technology (IT) (extended team)	• Providing support and maintenance for the tools and automation infrastructure used by the compliance program • Creating and/or acquiring new tools based on OSRB requests	Internal open-source compliance training	As part of their role
Software Team Managers	• Ensuring that developers follow open-source compliance policy and guidelines • Authorizing developers contribution to third party open-source projects during paid time • Identifying relevant issues that need to be discussed with the OSRB (license choice for new projects, integration of new third party open-source components) and ensure that indications from the OSRB are followed by developers	Internal open-source compliance training	As part of their role
Software Developers	Contributing to open-source software projects they have been assigned to (including third party projects) following the Open Source Compliance Policy and Implementation Guidelines	Internal open-source compliance training	As part of their role

All participants in the Program (including Software Staff) must undertake training covering the competencies required for their role, and at a minimum basic training. You can find details of the training requirements for each role in the training requirement page of the Appendix to these Guidelines. Such training should be held every 12 months unless there is a major update to the Policy or to any implementation guideline so that it may be triggered more frequently.

Every member of Software Staff in the Program will be assessed and records of the assessment will be found in the learning platform indicated in the Appendix.

Awareness

Open Source Compliance objectives are set out in the introduction to these Policy Implementation Guidelines. It is important that Oniro Project Working Group adheres to the Open Source Policy and to these Guidelines. Failure to do so may lead to:

- legal claims from the holders of copyright or other intellectual property rights in code we use
- jeopardizing the relationship with developer and user communities, and losing their support
- claims from customers
- inadvertent release of proprietary code
- loss or reputation
- loss of revenue
- breach of contract with suppliers and customers

The company training and assessment program will cover the objectives of each Program in which one participates, their role within the Program, and implications to the Company and to individuals for non-conformance. Evidence of the assessment can be found in the learning platform indicated in the Appendix.

Program Scope

These Open Source Policy Implementation Guidelines cover and apply to all open source software developed and made available by Oniro Project Working Group, including all activities related to the development, improvement, testing and release of Oniro Project and related projects.

License Obligations

Obligations, restrictions and rights granted by most prevalent open source Licenses (as those listed in 'Choose a License' website) are reviewed and documented by the Audit Team using functionalities provided by their license scanning tool, and are included in each software component's internal report generated through such tool.

As for uncommon open source licenses (and for uncommon variants of common licenses), an assessment by the Legal Team is required, which will be managed through the OSS issue tracker described in the Appendix (the "OSS Issue Tracker"); issues in this respect should be opened by any role who encounters an uncommon Identified License that appears not to have been reviewed yet in the OSS Issue Tracker or in the OSS wiki described in the Appendix (the "OSS Wiki"). The final outcome of the assessment will be included in the database of the license scanning tool by the Audit Team.

Oniro Project projects are generally intended to be released as source only distributions. Distribution of application and/or library binaries should be generally avoided within Oniro Project projects; however, some binary blobs could have to be distributed along with Oniro Project, to enable compatibility with certain hardware devices and components. License obligations, restrictions and rights related to such binary blobs shall be always reviewed and assessed by the Legal Team.

Since Oniro Project is intended to be implemented downstream by device makers, who will typically perform a binary distribution of modified parts of such software, Oniro Project Working Group commits to provide them with some basic assessment and information on license obligations, restrictions in the context of a typical binary/firmware distribution, including information about the existence and the license conditions of possible third party binary blobs.

In case of binary/firmware/proprietary software distribution, this possibility must be clearly identified in the top level documentation of Oniro Project.

In reviewing and documenting the obligations, restrictions and rights granted by each Identified License, Oniro Project Working Group commits to make use of, and to contribute to, open source resources such as 'Choose a License'.

Relevant Tasks Defined and Supported

External Open Source Inquiries

Anyone receiving an Open Source Compliance inquiry from outside the Oniro Project Working Group shall refer to the OSRB, which shall have overall responsibility for dealing with the inquiry, and – where appropriate – for assigning the handling of all or part of it to the appropriate role(s) within the company. This process will be managed through the OSS Issue Tracker.

In all public documentation concerning Oniro Project it should be stated that all external inquires concerning open source licensing should be sent the following email address: davide.ricci@huawei.com. Email messages sent to such email address will automatically trigger the creation of an issue on the OSS Issue Tracker, which will be assigned to OSRB members who will handle it.

Resourcing

To ensure that all tasks in the Program are executed and effectively resourced, roles, responsibilities and time requirements are set forth in sec. 3.1.2.

The Compliance Officer is responsible for ensuring that adequate funding is allocated for the success and the effectiveness of the Program.

The main Open Source policy and these implementation guidelines are open to constant review and update in our git server described in the Appendix (the "Internal Git Server"). Issues about modifications and proposed enhancements should be opened in the OSS Issue Tracker.

Legal expertise from Legal Team pertaining to Open Source Compliance is accessible to any role through the OSS Issue Tracker. Before opening an issue with the Legal Team, please check in the OSS Issue Tracker and in the OSS Wiki if the issue has been already addressed before.

Any other open source compliance issue will be managed through the OSS Issue Tracker and assigned to the OSRB.

Open Source Content Review and Approval

Bill of Materials

Our process for creating and managing bill of materials that includes each Open Source component (and its Identified License) distributed within the Supplied Software is based on an open source compliance software CI/CD toolchain described in the Appendix. It is intended to be a continuous process overseen by the OSRB and carried out by the Audit Team and by the Legal Team. During software development process, project bill of materials is gradually built, updated and validated, so that at release time the the toolchain should be able to automatically generate a complete BOM by reusing human validation work done during development.

License Compliance

In the CI/CD process described at sec. 3.3.1, generally any potential legal issue should be tracked and addressed since the very beginning of software development. Software Developers, Software Team Managers, Audit Team members are required to check, whenever a new component is added to a project, or an existing third party component is modified: *(i)* what is the license attached to it, *(ii)* if there is any information concerning that component and/or its license in the OSS Wiki or in the OSS Issue Tracker; and, if they spot a potential issue or they have doubts, they shall open an issue in the OSS Issue Tracker, which will be handled by the Legal Team.

Compliance Artifacts Creation and Delivery

Compliance Artifacts

The process outlined in sec. 3.3.1 allows the creation of Compliance Artifacts with automated CI/CD pipelines on our Internal Git Server – namely, a SPDX document for each software package/component and a SPDX document describing the whole project distributions, and an internal report covering internal information and legal assessment for each component.

Open Source Community Engagements

Contributions To Other Projects

General principles and rules covering contribution to third party open source projects are covered by our main Open Source Policy.

Contributions to third parties projects must be forked in the official account of Oniro Project Working Group and subject to the internal procedures to open repositories, forks, etc. as updated from time to time on the OSS Wiki.

Local repositories must me cloned on the internal IT services that must have been expressly cleared for this use and are approved by the Compliance Officer. Appointment to follow an external project **must not hinder compliance** with the IT security procedures in place.

External Contributions From Others To Our Projects

As to external contributions *to* Oniro Project Working Group's open source project, each repository must have a `contributing.md` file that reflects this policy. All contributions that do not contain a sign-off statement will be rejected or will receive a request to sign-off.

This must be made an **automated** process as much as possible, integrated in the CI/CD environment of all processes hosted or initiated by Oniro Project Working Group.

6.2.4 Copyright Notice.

Authors: Carlo Piana and Alberto Pianon (Array)

Reviewers: Davide Ricci and Christian Paterson (Huawei)

6.2.5 Appendix.

This Appendix describes the internal processes, document, resources necessary to follow the Implementation Guidelines.

Note: this document is a stub for users. It could contain actual content or just point to external resources.

Appendix.1. General List of useful contacts

Appendix.2. Training

Appendix.2.1. Training Requirements for Each Role

Appendix.2.2. Learning Platform

Platform delivers training and keeps track of undertaken and training not undertaken by each individual.

Appendix.3. Issue Tracker

Appendix.4. OSS Wiki

Appendix.5. Description of integration of compliance tools in CI/CD

CHAPTER

SEVEN

SECURITY POLICIES

This section describes the security policies and practices of Oniro Project.

7.1 Vulnerability Handling Process (draft)

Oniro Project aims to build a secure system from the foundation, applying the best industry practices in terms of development quality. However, as in every software project, bugs do happen. Some of them will offer a possibility to be exploited by an attacker and are called security vulnerabilities. This process explains how we handle security issues and extends the more generic bug handling process.

We work in the open, including the process of handling security issues. To protect deployed products, sometimes we need to delay releasing information related to security issues, following the industry best practices. However, all information about vulnerabilities is becoming publicly available at the end.

7.1.1 How to Report a Vulnerability?

If you think you have found a security issue in our distribution, please contact us immediatelly by posting a confidential issue in our bug tracker in a dedicated security project.

To do so, login into our issue tracker or create a new account if you do not have one yet. Click on New issue, then make sure to check the checkbox at the bottom 'This issue is confidential and should only be visible to team members with at least Reporter access'. Please use the 'Issue' type of ticket and the associated template. Fill in the title, answer the questions in the 'Description' field. Then click 'Create issue'.

Your report should contain a description of the issue, the steps you took to reproduce the issue (including the image name), affected versions, and, if known, any mitigations for the issue.

We plan to add a security-related mailing list and a possibility to send GPG-encrypted email in the near future.

We aim to acknowledge the reception within one working day, and responding with a first assessment within three working days. We follow a 90 days disclosure timeline.

We will be happy to acknowledge your work in the vulnerability announcement, and will do so if you do not object.

This first section is included in the `SECURITY.md` file in our high-level project repositories.

We use responsible vulnerability disclosure, and you can read more about this kind of disclosures in the Vulnerability Disclosure Cheat Sheet from OWASP or the detailed CERT Guide to Coordinated Vulnerability Disclosure .

7.1.2 Security Response Team (SRT)

Our Security Response Team (SRT) is reviewing reported security issues and updating the security policies. Members of the team are chosen by the project partners and elected by and from the project developers. Ideally, they should have security experience. The SRT has a minimum of two members.

The SRT may decide the reported issue is indeed a security vulnerability (with assigned severity), a non-confidential bug, a feature request, or the feature is working as expected. The team notifies the reporter of the decision and provides explanations. If the issue is classified as a bug, the team converts it to a normal bug. If it is a feature request, the team asks the reporter to create a feature request and closes the issue. If the feature is working as expected, the team closes the security issue. The SRT also sets up the issue domain (for example compiler, base system etc).

The SRT also makes an initial decision if the issue is in the code maintained by the projects (issues where we are upstream) or maintained outside the project (issues where we are downstream). This decision can be changed later if new information becomes available.

The SRT meets weekly on a status meeting and participates in the general Bug triage/prioritization meeting.

7.1.3 Classification of Issues

Security issues are classified as high, medium, and low severity. As a rule of thumb, we map the Base CVSS score from v3.1 in the following way:

- 0 to 3.9 - low severity
- 4.0 to 6.9 - medium severity
- 7.0 and above - high severity

7.1.4 When the Issue is in the Code Maintained by the Project

When the source code where the issue comes from is maintained by the Project, the SRT creates a confidential ticket about the issue and assigns it to the relevant developer. The security team also verifies which versions are affected.

If the security team judges it could be exploited, they request a CVE number for the issue and set up the embargo duration. It is by default 90 days, and may be different if necessary (for example, if the fix will be complicated to deploy, or the issue will be known earlier for some reasons).

The CVE number is mentioned in the confidential ticket, but should not be used in any other communication until the end of the embargo. The commit messages and documentation should be stating what was fixed (a NULL pointer, a missing lock, etc).

The fix should be developed in a private repository and the reporter may be taking part in the development if they wish so.

When the fix is available, it should be included in the main branch and backported to the release branches. If the issue is of 'high' severity, an immediate bugfix release should be produced. If it is a 'medium' or 'low' severity, the fix waits until the next regular bugfix release. In the case of a critical issue, the security team together with the release team may decide in distributing patches to the affected users.

7.1.5 Handling Upstream Security Issues

If the issue was identified in upstream code, we do report an upstream security issue using the upstream project's process. We track the investigation status and the fix in our bug tracking system. When a fix is available, we do an update of the affected source, with backporting if necessary.

If the upstream project does not respond, or does respond very slowly, we may decide to develop a patch on our own. In this case, the vulnerability is using the process for issues where we are upstream.

7.1.6 Detailed Workflow

Our process contains four phases: monitoring, assessment, remedy, and notification.

Monitor

We actively monitor the ecosystem for potential security issues in the code developed by us, and in the code we distribute. This includes monitoring the official CVE list and other vulnerability databases, running code analysis tools, monitoring related blog posts or conference presentations. In addition to that, a regular bug might be marked as a potential security issue. If a potential issue appears, any project member (or an external observer) may fill in a security issue.

As we depend on much upstream code, we also monitor specific mailing lists informing about security issues in those projects, including special notification lists for issues under embargo.

This step has no equivalent in our Bug policy.

Assess

When we learn about a potential security issue, we start by acknowledging the information.

If the issue comes from a CVE database, we verify if we are affected by the vulnerability at all (for example, we are not affected by the software we do not include directly, nor by a dependency).

The SRT reproduces the issue during the assessment process and documents the needed steps, including configuration details (like package versions), system (like the processor architecture), and commands used.

The SRT declares a security issue if it compromises one or more of the three features: avaliability, integrity, or confidentiality.

When assessing an issue, the SRT may confirm it is a security issue or decide it is a regular bug. The team may also decide that a feature is missing or it behaves as intentionally designed and specified.

In all cases, the SRT notifies the reporter of the assessment.

Our aim is to acknowledge the reception within one working day, and respond with a first assessment within three working days.

This step is an equivalent of the Triage and Prioritize steps of the Bug process.

Remedy

When the issue is confirmed as a security issue, the process of developing a fix begins. The reporter may be included in the process if they wish so. The SRT also applies for a CVE issue number and decides if there will be an embargoed notification before the public release.

The SRT notifies the developers who should know about the issue and who should develop the fix. The communication happens over a private channel.

Developers create a patch and associated test cases in a private branch. They also backport the fix to supported releases. In the case of non-public issues, the developer should mention in the patch description only what is fixed, not include any reference to the CVE. A fix might have a title like 'fix a crash in module X' or 'add a missing unlock in module Y'.

They also prepare the release for issues with 'high' severity. 'Medium' and 'low' severity issues are fixed in regular bugfix releases.

We follow the rules of the upstream projects, if applicable.

This step is an equivalent of the Fix step of the Bug process.

Notify

If an embargoed notification happens, it is sent between 5 to 30 days before the expected publication date. The actual timeframe depends on the situation and affected parties. For example, if deployed devices are affected, the SRT may choose a longer time to allow patching of the vulnerable devices. The embargoed notification includes the CVE identification number, description of the issue, affected versions, the patch itself and the way it will be distributed, the public disclosure date, and the reporter credits. The SRT monitors the responses to the notification messages to fix any outstanding issues.

When the issue enters this phase, all documentation of the issue needs to be ready. The SRT and developers prepare a security advisory (if appropriate), the information for the release notes and the release announcement.

This step (with the Publish one described below) is an equivalent of the Release step of the Bug process.

Publish

The publication step consits of releasing the information about the issue publicly. The information prepared earlier is published on the public disclosure date. The SRT updates the CVE information.

The release notes contains a list of all vulnerabilities fixed in the release. For issues with important impact, the SRT might decide on a dedicated advisory.

This step (with the Notify one described above) is an equivalent of the Release step of the Bug process.

Glossary

- CVE (Common Vulnerabilities and Exposures) - a common system for vulnerability naming and referencing. https://en.wikipedia.org/wiki/Common_Vulnerabilities_and_Exposures
- CVSS (Common Vulnerability Score System) - a score standard for security vulnerabilities, ranging from 0.0 (no impact) to 10.0 (critical impact). https://en.wikipedia.org/wiki/Common_Vulnerability_Scoring_System

Acknowledgements

This process was inspired by the OSS vulnerability guide, the OpenSSF Vulnerability Disclosure WG guide to disclosure for OSS projects, other work from the OpenSSF vulnerability-disclosures WG, Zephyr project security policy.

7.2 Security Practices

Oniro Project aims to build a secure system foundation, applying the best industry practices in terms of development quality. This page describes the security-related settings included in the distribution.

7.2.1 Linux Kernel Hardening Options (Qemu Builds Only)

Hardening is a process of securing the system by reducing the attack surface for vulnerabilities, such as removing unused or unsafe options and modules, for example, setting up the defaults considered to be safe, and enabling additional checks. The goal of hardening is to make the attack harder or reduce its impact.

Hardening options improve security and allow developers to detect bugs early thus increasing software quality in general.

Hardening and security options *may* have performance costs (e.g. due to additional checks in the code path). Oniro attempts to assure that security hardening features do not increase overhead more than %5, in general.

On the other hand, additional tests come with an additional computing costs and may reduce performance. The hardening options suggested are checked against a research from a benchmark[1], and it is verified that only such options are enabled, that do not raise the computation cost, and the performance loss is expected to be around 5 percent at most.

The Linux kernel hardening options of Oniro Project are descibed as follows, and suggested hardening options are defined in configuration files in `oniro/meta-oniro-core/recipes-kernel/linux/linux/`.

You can find the decisions that have been made for different categories of options as follows, and the descriptions are taken from the Kconfig with changes for better readability:

Memory Allocator

The hardening options of memory allocation protect against issues like leaking data freed from memory, and accessing wrong memory zones.

Source files: `oniro/meta-oniro-core/recipes-kernel/linux/linux/hardening_allocator.cfg` and `oniro/meta-oniro-core/recipes-kernel/linux/linux/hardening_allocator_perf.cfg`.

Config option	Oniro state
CONFIG_SLAB_FREELIST_RANDOM CONFIG_SLAB_FREELIST_HARDENED CONFIG_SHUFFLE_PAGE_ALLOCATOR CONFIG_PAGE_POISONING CONFIG_PAGE_POISONING_NO_SANITY CONFIG_PAGE_POISONING_ZERO CONFIG_INIT_ON_ALLOC_DEFAULT_ON	On On On On On On On

CONFIG_SLAB_FREELIST_RANDOM=y

Description: Randomize the freelist order used on creating new pages. This security feature reduces the predictability of the kernel slab allocator against heap overflows.

[1] Benchmarking of Linux kernel security options - a blog post from BlackIkeEagle Kconfig hardening tests

Recommendation source: KSPP[2]

CONFIG_SLAB_FREELIST_HARDENED=y

Description: Many kernel heap attacks try to target slab cache metadata and other infrastructure. This option makes minor performance sacrifices to harden the kernel slab allocator against common freelist exploit methods. Some slab implementations have more sanity-checking than others. This option is most effective with SLUB.

Recommendation source: KSPP[2]

CONFIG_SHUFFLE_PAGE_ALLOCATOR=y

Description: Randomization of the page allocator improves the average utilization of a direct-mapped memory-side-cache. See section 5.2.27 Heterogeneous Memory Attribute Table (HMAT) in the ACPI 6.2a specification for an example of how a platform advertises the presence of a memory-side-cache. There are also incidental security benefits as it reduces the predictability of page allocations to compliment SLAB_FREELIST_RANDOM, but the default granularity of shuffling on the `MAX_ORDER - 1` i.e, 10th order of pages is selected based on cache utilization benefits on x86.

While the randomization improves cache utilization, it may negatively impact workloads on platforms without a cache. For this reason, by default, the randomization is enabled only after runtime detection of a direct-mapped memory-side-cache. Otherwise, the randomization may be force enabled with the `page_alloc.shuffle` kernel command line parameter.

Recommendation source: KSPP[2]

CONFIG_PAGE_POISONING=y

Description: Fill the pages with poison patterns after `free_pages()` and verify the patterns before `alloc_pages`. The filling of the memory helps reduce the risk of information leaks from freed data. This does have a potential performance impact if enabled with the `page_poison=1` kernel boot option.

Note that "poison" here is not the same thing as the "HWPoison" for MEMORY_FAILURE. This is software poisoning only.

Recommendation source: KSPP[2]

CONFIG_PAGE_POISONING_NO_SANITY=y

Description: Skip the sanity checking on alloc, only fill the pages with poison on free. This reduces some of the overhead of the poisoning feature.

Recommendation source: KSPP[2]

CONFIG_PAGE_POISONING_ZERO=y

Description: Instead of using the existing poison value, fill the pages with zeros. This makes it harder to detect when errors are occurring due to sanitization, but the zeroing at free means that it is no longer necessary to write zeros when `GFP_ZERO` is used on allocation.

Recommendation source: KSPP[2]

CONFIG_INIT_ON_ALLOC_DEFAULT_ON=y

Description: This has the effect of setting `init_on_alloc=1` on the kernel command line. This can be disabled with `init_on_alloc=0`. When `init_on_alloc` is enabled, all page allocator and slab allocator memory will be zeroed when allocated, eliminating many kinds of "uninitialized heap memory" flaws, especially heap content exposures. The performance impact varies by workload, but most cases see <1% impact. Some synthetic workloads have measured as high as 7%.

Recommendation source: KSPP[2]

[2] Kernel Self Protection Project (KSPP) is an initiative that works on adding security features to the Linux kernel. Among other things, they previde a set of recommendations on options to use.

Reducing Attack Surface

The following options remove some obsolete or un-needed features, which could make attacks easier:

Config option	Oniro state
CONFIG_COMPAT_BRK CONFIG_PROC_KCORE CONFIG_BINFMT_MISC	Off Off Off

Option: CONFIG_COMPAT_BRK is not set

Reason: Disabling this option enables heap randomization. Randomization of the heap has been introduced around the year 2000, we do not expect Oniro to run binaries compiled before that date, so we can enable.

Description: Randomizing heap placement makes heap exploits harder, but it also breaks ancient binaries (including anything libc5 based). This option changes the bootup default to heap randomization disabled, and can be overridden at runtime by setting `/proc/sys/kernel/randomize_va_space` to 2.

Recommendation source: KSPP[Page 104, 2]

Option: CONFIG_PROC_KCORE is not set

Description: Provides a virtual ELF core file of the live kernel. This can be read with gdb and other ELF tools. No modifications can be made using this mechanism.

Recommendation source: KSPP[Page 104, 2]

Option: CONFIG_BINFMT_MISC is not set

Description: If you say `Y` here, it will be possible to plug wrapper-driven binary formats into the kernel. You will like this especially when you use programs that need an interpreter to run like Java, Python, .NET or Emacs-Lisp. It is also useful if you often run DOS executables under the Linux DOS emulator DOSEMU (read the DOSEMU-HOWTO). Once you have registered such a binary class with the kernel, you can start one of those programs simply by typing in its name at a shell prompt; Linux will automatically feed it to the correct interpreter.

Recommendation source: KSPP[Page 104, 2]

Dmesg Options

Those options are related to the kernel log in dmesg:

Config option	Oniro state
CONFIG_SECURITY_DMESG_RESTRICT	On

Source files: `oniro/meta-oniro-core/recipes-kernel/linux/linux/hardening_dmesg.cfg`

CONFIG_SECURITY_DMESG_RESTRICT=y

Description: This enforces restrictions on unprivileged users reading the kernel syslog via dmesg(8).

If this option is not selected, no restrictions will be enforced unless the `dmesg_restrict` sysctl is explicitly set to (1).

Recommendation source: KSPP[Page 104, 2]

Compiler-level Hardening

Those options enable checks done by the compiler:

Config option	Oniro state
CONFIG_FORTIFY_SOURCE	On

Source file: `oniro/meta-oniro-core/recipes-kernel/linux/linux/hardening_fortify_source.cfg`.

CONFIG_FORTIFY_SOURCE=y

Description: Detect overflows of buffers in common string and memory functions where the compiler can determine and validate the buffer sizes.

Recommendation source: KSPP[Page 104, 2]

Memory Accesses

With those options we disable the complete physical memory access and detect unsafe memory permissions:

Config option	Oniro state
CONFIG_DEBUG_WX CONFIG_DEVMEM	On Off

Source file: `oniro/meta-oniro-core/recipes-kernel/linux/linux/hardening_memory.cfg`.

CONFIG_DEBUG_WX=y

Description: Generates a warning if any `W+X` mappings are found at boot.

This is useful for discovering cases where the kernel is leaving `W+X` mappings after applying `NX`, as such mappings are a security risk.

Look for a message in dmesg output like this:

`/mm:  Checked W+X mappings:  passed, no W+X pages found.`

or like this, if the check failed:

`/mm:  Checked W+X mappings:  failed, W+X pages found.`

Note that even if the check fails, your kernel is possibly still fine, as `W+X` mappings are not a security hole in themselves, what they do is that they make the exploitation of other unfixed kernel bugs easier.

Recommendation source: KSPP[Page 104, 2]

CONFIG_DEVMEM is not set

Reason: Disabling access to the whole memory mapping.

Description: Say `Y` here if you want to support the `/dev/mem` device. The `/dev/mem` device is used to access areas of physical memory.

Recommendation source: KSPP[Page 104, 2]

Copying from Userspace

Those options add verification when copying potentially malicious data from the user space:

Config option	Oniro state
CONFIG_HARDENED_USERCOPY CONFIG_HARDENED_USERCOPY_FALLBACK	On Off

File: `oniro/meta-oniro-core/recipes-kernel/linux/linux/hardening_usercopy.cfg`.

Reason: Perform boundary checks on memory when copying to/from the kernel. Also disable the whitelisting with this check.

CONFIG_HARDENED_USERCOPY=y

Description: This option checks for obviously wrong memory regions when copying memory to/from the kernel (via `copy_to_user()` and `copy_from_user()` functions) by rejecting memory ranges that are larger than the specified heap object, span multiple separately allocated pages, are not on the process stack, or are part of the kernel text. This kills entire classes of heap overflow exploits and similar kernel memory exposures.

Recommendation source: KSPP[Page 104, 2]

CONFIG_HARDENED_USERCOPY_FALLBACK is not set

Reason: Do not enable the whitelisting for usercopy checks. The kernel keeps a list of memory zones allowed for copying user data. This option was added to allow the transition from around 2017, but we assume all code we use has been already fixed.

Description: This is a temporary option that allows missing usercopy whitelists to be discovered via a `WARN()` to the kernel log, instead of rejecting the copy, falling back to non-whitelisted hardened usercopy that checks the slab allocation size instead of the whitelist size. This option will be removed once it seems like all missing usercopy whitelists have been identified and fixed. Booting with `slab_common.usercopy_fallback=Y/N` can change this setting.

Recommendation source: KSPP[Page 104, 2]

Data Validation

With those options we add verification of the internal kernel data structures:

Config option	Oniro state
CONFIG_DEBUG_NOTIFIERS CONFIG_DEBUG_LIST CONFIG_DEBUG_SG CONFIG_BUG_ON_DATA_CORRUPTION CONFIG_SCHED_STACK_END_CHECK	On On On On On

File: `oniro/meta-oniro-core/recipes-kernel/linux/linux/hardening_validation_checks.cfg`.

CONFIG_DEBUG_KERNEL=y

Reason: Needed for `CONFIG_SCHED_STACK_END_CHECK`.

Description: Say `Y` here if you are developing drivers or trying to debug and identify kernel problems.

Recommendation source: KSPP[Page 104, 2]

CONFIG_DEBUG_NOTIFIERS=y

Reason: Perform additional checks for the notifier call chains.

Description: Enable this to turn on sanity checking for notifier call chains. This is most useful for kernel developers to make sure that modules properly unregister themselves from notifier chains. This is a relatively cheap check, but if you care about maximum performance say `N`.

Recommendation source: KSPP[Page 104, 2]

CONFIG_DEBUG_LIST=y

Reason: Perform additional checks for the lists.

Description: Enable this option to turn on extended checks in the linked-list walking routines.

Recommendation source: KSPP[Page 104, 2]

CONFIG_DEBUG_SG=y

Reason: Perform additional checks for scatter-gather lists.

Description: Enable this option to turn on checks on scatter-gather tables. This can help find problems with drivers that do not properly initialize their sg tables.

Recommendation source: KSPP[Page 104, 2]

CONFIG_BUG_ON_DATA_CORRUPTION=y

Reason: Detect data corruptions early and stop with a BUG() error message.

Description: Select this option if the kernel should `BUG()` when it encounters data corruption in kernel memory structures when they get checked for validity.

Recommendation source: KSPP[Page 104, 2]

CONFIG_SCHED_STACK_END_CHECK=y

Reason: Check stack overflow when calling `schedule()`. If it happens, stop the system with a `panic` as the memory region could not be trusted.

Description: This option checks for a stack overrun on calls to `schedule()`. If the stack end location is found to be over written, always panic as the content of the corrupted region can no longer be trusted. This is to ensure no erroneous behaviour occurs which could result in data corruption or a sporadic crash at a later stage once the region is examined. The runtime overhead introduced is minimal.

Recommendation source: KSPP[Page 104, 2]

Options not Applied (yet)

GCC plugins

GCC plugins offer ways to additionally harden the code at the compiler level. These options are proposed to be applied in the future releases.

IOMMU

IOMMU is not enabled yet.

Panic on Oops

File: Source file: `hardening_fortify_source.cfg`

KSPP[Page 104, 2] recomends setting up the following: ` CONFIG_PANIC_ON_OOPS=y CONFIG_PANIC_TIMEOUT=-1 `

They cause the kernel to reboot on serious error (Oops, see `the Oops Wikipedia page <https://en.wikipedia.org/wiki/Linux_kernel_oops>` for more information).

This might happen during development and result in a reboot loop, so it was decideed not to enable this options in the development phase. You can add this file to the kernel configuration if it is safe in your product.

Module Signing

Module signing is not enabled yet, we need the key infrastructure set up.

Known Issues

None.

7.2.2 References

CHAPTER

EIGHT

CODE OF CONDUCT

The Oniro Project community complies with the code of conduct specified in Contributor Covenant. of the open source community. For details, see https://www.contributor-covenant.org/version/1/4/code-of-conduct/

8.1 Our Pledge

In the interest of fostering an open and welcoming environment, we as contributors and maintainers pledge to make participation in our project and our community a harassment-free experience for everyone, regardless of age, body size, disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, religion, or sexual identity and orientation.

8.2 Our Standards

Examples of behavior that contributes to creating a positive environment include:

- Using welcoming and inclusive language
- Being respectful of differing viewpoints and experiences
- Gracefully accepting constructive criticism
- Focusing on what is best for the community
- Showing empathy towards other community members

Examples of unacceptable behavior by participants include:

- The use of sexualized language or imagery and unwelcome sexual attention or advances
- Trolling, insulting/derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or electronic address, without explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

8.3 Our Responsibilities

Project maintainers are responsible for clarifying the standards of acceptable behavior and are expected to take appropriate and fair corrective action in response to any instances of unacceptable behavior.

Project maintainers have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, or to ban temporarily or permanently any contributor for other behaviors that they deem inappropriate, threatening, offensive, or harmful.

8.4 Scope

This Code of Conduct applies within all project spaces, and it also applies when an individual is representing the project or its community in public spaces.

Examples of representing a project or community include using an official project e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event.

Representation of a project may be further defined and clarified by project maintainers.

8.5 Implementation

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported by contacting the project team.

All complaints will be reviewed and investigated and will result in a response that is deemed necessary and appropriate to the circumstances. The project team is obligated to maintain confidentiality with regard to the reporter of an incident. Further details of specific enforcement policies may be posted separately.

Project maintainers who do not follow or enforce the Code of Conduct in good faith may face temporary or permanent repercussions as determined by other members of the project's leadership.

CHAPTER

NINE

COMMUNITY CHAT PLATFORM

Contents

9.1 Overview

The Oniro Project project primarily uses a chatroom for community discussions, everyone is welcome! If you are looking for help or interested in contributing to the project, feel free to interact with maintainers, contributors and other community members.

Project's chatroom is hosted in Libera IRC Network. which is reachable using IRC protocol. IRC Channel #oniroproject can be joined using any IRC client by connecting to this address:

- irc://libera.chat/#oniroproject.

Alternatively for user's convenience third party services can also be used from web (terms of service apply):

- <https://web.libera.chat/#oniroproject>
- <https://web.libera.chat/gamja/#oniroproject>
- <https://kiwiirc.com/client/irc.libera.chat/#oniroproject>

The user might prefer to install an IRC client to connect to #oniroproject channel, like:

- HexChat (Native IRC client for Linux, Windows)

More details are explained on the Libera website.

Another possibility to use this IRC room is to use our Matrix bridge to IRC. One benefit is that everyone can catch up by reading existing logs on reconnection.

CHAPTER

TEN

RELEASES

This section contains information related to releases of Oniro Project.

10.1 Aladeen - 0.1.0

10.1.1 About

The objective of this document is to provide basic introductory information about included functionalities, known issues, instructions guidance for the Aladeen release of the Oniro Project project.

The Oniro Project project is meant to serve as a solid base foundation for products. It is not a standalone product itelf, but rather a starting project for other projects and products.

10.1.2 Scope

Release codename: Aladeen

Release version: 0.1.0

Release timeframe: 2020/11/15 .. 2021/04/12

The Objectives of the Release

The purpose of this release is to provide the European Oniro Project project with a set of initial functionalities after the first code release in September, 2020. This April release serves as a solid foundation upon which we can share the entire 2021 roadmap of the project.

More details on the Oniro Project goals can be found in the *Oniro Project Vision and Aims* document.

The List of Software Features Included

Functionalities	Area
Bitbake build infrastructure	Build tools
Continuous IP compliance and Openchain IP policy	FOSS compliance
Linux 5.10	Kernel
Zephyr 2.5	Kernel
FreeRTOS (experimental)	Kernel
GCC / LLVM cross toolchains	Toolchain
gdb / gdbserver debugging	Tools
Qemu targets for Intel and ARM	Simulators
Build flavors and base / extra root fs images	Build tools and root fs
Smart Panel Blueprint	Reference Blueprint
Continuous IP compliance and Openchain IP policy	FOSS compliance
Release Bill of Material and FOSS compliance dashboard	FOSS compliance
Application Compatibility Test Suite	Testing and compatibility
CI / CD pipelines and DevSecOps public cloud infrastructure	DevSecOps
Web portal and communication assets including Twitch channel	Marcom
Automated sphinx-based documentation pipeline	Documentation

Supported Hardware Platforms

Board (chipset)	Supported kernels	Board documentation
Nitrogen (nRF52832 - Cortex-M4)	Zephyr	*96Boards Nitrogen*
Avenger96 (STM32 - Cortex-A7 & Cortex-M4)	Linux & Zephyr	*96Boards Avenger96*
SBC-C61 (NXP i.MX 8M - Coretex-A53 & Cortex M4)	Linux	*SBC-C61 SECO*
SBC-B68-eNUC (Intel x86)	Linux	*SBC-B68-eNUC SECO*

Test Report

Not available for this release.

10.1.3 Installation

Quick Build provides an example of how to build the Oniro Project project for a supported target. Visit the *Hardware Support* section for instructions on how to build for other supported targets.

Due to project rename and reorganization of the repos Aladen release doesn't require repo tool. Entire workspace is archived into sources tarball and can be prepared with:

```
$ tar -xvf asos-0.1.0.tar.xz
$ cd asos-0.1.0
```

Known Issues

The Avenger96 target (stm32mp1-av96) does not ship with firmware files for Bluetooth, Wifi and the image processor by default. We are working on a solution to re-distribute them during the normal image builds for this board.

Source Code Available

Source code and GPG signatures:

asos-0.1.0.tar.xz

asos-0.1.0.tar.xz.sig

The release is signed with the key:

```
-----BEGIN PGP PUBLIC KEY BLOCK-----

mDMEYMoZMBYJKwYBBAHaRw8BAQdAJhMs58bN6YYu2v9xZcyIkSSKQdpRm61Cac7y
FPuN8BK0VlN0ZWZhbiBTY2htaWR0IChBbGwgU2NlbmFyaW9zIE9TIFJlbGVhc2Ug
U2lnbmluZyBLZXkgMjAyMCkgPHN0ZWZhbi5zY2htaWR0QGh1YXdlaS5jb20+iJYE
ExYIAD4WIQQHOpqTQvKyQyHYvxd7YfaxNxtJNwUCYMoZMAIbAwUJAeEzgAULCQgH
AgYVCgkICwIEFgIDAQIeAQIXgAAKCRB7YfaxNxtJN1DDAP4q1a4jJh0PVtF/OTRf
AhbBb2R8pFx5WLTGgUBvyFNXvwD9HFZHJCH7z8OvcO+70r27J3Kbwn/COPsu4eZI
x11UYwu4MwRgyhkwFgkrBgEEAdpHDwEBB0CBT/QrS+DquNFyzLFiwntmiz3ml553
WAFapVYBooBY2Yh+BBgWCAAmFiEEBzqak0LyskMh2L8Xe2H2sTcbSTcFAmDKGTAC
GyAFCQHhM4AACgkQe2H2sTcbSTdRCwD/cstokK68pKzdpVldJmAth54jQu6wJtEJ
4h5ABbPs92gBAPMfzKk15b1c5GaXOCrduOx8EPX4okH9yE+v0UfESGkBuDgEYMoZ
MBIKKwYBBAGXVQEFAQEHQH/9Ag5C0I+DL4zu09dT6Gp/Fgzvg5dTqxSLf/5TlBEd
AwEIB4h+BBgWCAAmFiEEBzqak0LyskMh2L8Xe2H2sTcbSTcFAmDKGTACGwwFCQHh
M4AACgkQe2H2sTcbSTesYAD+O0A0NMi6OnqO7Ozknpzf/+FgT5n/fYmbc1dbAzSp
klgA/jeBSvrxjZ1V5QoptpWamjhVwbdWavCR5iDGcVa9aCkG
=lKOe
-----END PGP PUBLIC KEY BLOCK-----
```

10.1.4 Devops Infrastructure

To learn more about our approach to CI (Continuous Integration) strategy used for this release, please see:

Continuous Integration document

10.1.5 Contributions

If you are a developer eager to know more details about Oniro Project or just an enthusiast with a patch proposal, you are welcome to participate to our Oniro Project ecosystem development. To do so, please sign-up using the process described below:

Contributing to Oniro Project document

10.1.6 License

Project manifest as well as project-specific meta-layers, recipes and software packages are, unless specified otherwise, published under Apache 2.0 license. The whole operating system built by users from the project manifest is an aggregate comprised of many third-party components or component groups, each subject to its own license conditions.

Official project release includes only the project manifest as well as project-specific meta-layers, recipes. Any reference binary image, build cache and any other build artifacts are distributed as a convenience only and are not part of the release itself.

10.2 Jasmine - 1.0.0

10.2.1 About

The objective of this document is to provide basic introductory information about included functionalities, known issues, instructions guidance for the Jasmine release of the Oniro Project project.

The Oniro Project project serves as a solid base foundation for products. It is not a standalone product itself but rather a starting project for other projects and products.

Within this release, we moved from the Incubation phase into the Developing phase. In practice, it means that Oniro Project has become a member project of the Eclipse Foundation. We believe that it enables us for further rapid development and extenstion in various markets.

10.2.2 Scope

Release codename: Jasmine

Release version: 1.0.0

Release timeframe: 2021-04-13 .. 2021-12-20

The Objectives of the Release

The objective of the *Jasmine* release is to prepare a solid ground for the Eclipse Foundation onboarding. That includes, but is not limited to:

- Project maturity gain
- Hardware & Software roadmap expansion
- IP compliance & policies deployment
- An extension of the available set of blueprints

The List of Software Features Included

- Raspberry Pi Hardware Enablement (REQ#253)
- Support and default to the 5.10.x LTS Linux Kernel (REQ#99)
- BSP: Add support for Nordic Semiconductor nRF52840 DK (REQ#248)
- Net: Overall IPv6 connectivity for the gateway (REQ#96)
- Net: Support open-source 802.11 stack on Linux OS (REQ#296)
- IP Compliance: Openchain 2.1 compliant IP policy (REQ#240)
- Experimental UI framework LVGL (REQ#279)

Please note that there is a set of blueprints available which could be built using this release. However, we can not guarantee that all newly developed blueprints will work with the previous releases of the Oniro Project.

In case of any doubts, please refer to a blueprint's documention or contact us using any of the communication channels available.

Supported Hardware Platforms

Board (chipset)	Supported kernels	Board documentation
Avenger96 (STM32 - Cortex-A7 & Cortex-M4)	Linux & Zephyr	*96Boards Avenger96*
Arduino Nano (nRF52840 - Cortex-M4)	Zephyr	*Arduino Nano 33 BLE*
Nitrogen (nRF52832 - Cortex-M4)	Zephyr	*96Boards Nitrogen*
nRF52840-DK (nRF52840 - Cortex-M4)	Zephyr	*nRF52840 DK*
RaspberryPI 4b (BCM2711)	Linux	*Raspberry Pi 4 Model B*
Seco SBC-B68-eNUC (Intel x86)	Linux	*SBC-B68-eNUC SECO*
Seco SBC-C61 (NXP i.MX 8M - Coretex-A53 & Cortex M4)	Linux	*SBC-C61 SECO*

Test Report

Not available for this release.

10.2.3 Installation

Quick Build provides an example of how to build the Oniro Project project for a supported target. Visit the *Hardware Support* section for instructions on how to build for other supported targets.

Visit *setting up a repo workspace* for instructions how to prepare workspace for development. Since Oniro Project uses repo tool for its development, you can use the release tag for the repo init commands as follows:

```
repo init -u https://booting.oniroproject.org/distro/oniro.git -b v1.0.0
```

Known Issues

1. The Avenger96 target (stm32mp1-av96) does not ship with firmware files for Bluetooth, WiFi, and the image processor by default. We are working on a solution to re-distribute them during the normal image build for this board.
2. Adding package_management feature breaks passwordless root login.
3. Smart Panel blueprint - Avenger96 HDMI Display problems.

Source Code Available

For more details on our repo structure, see:

OniroProject's GitLab document

The release is signed with the key:


```
-----BEGIN PGP PUBLIC KEY BLOCK-----
mDMEYaTV2hYJKwYBBAHaRw8BAQdAhCeM7i/qPIdJMs9SjBAYYsZ0lEhkJExfScVV
3PCAMNy0RU9uaXJvIFByb2plY3QgSmFzbWluZSBSZWxlYXNlIFNpZ25pbmcgS2V5
IDxzdGVmYW4uc2NobWlkdEBodWF3ZWkuY29tPoiWBBMWCAA+FiEEg6LprPlSFnVN
ljMIypzNhbi9XNYFAmGk1doCGwMFCQYGewAFCwkIBwIGFQoJCAsCBBYCAwECHgEC
F4AACgkQypzNhbi9XNa2UwD9El6WIkKBoZ/xAOdg/eUTe4ltzjyrMznGTjltuLS8
iq0A/iBrx+e9Jod5j3t9U5a99BTUWEe1hHi2lyAeGyCgvyoHuDgEYaTV2hIKKwYB
BAGXVQEFAQEHQJ+oeI2KvnvYtIixZc6uZGA6fhNFrBIQdVk65jU6/GcFAwEIB4h+
BBgWCAAmFiEEg6LprPlSFnVNljMIypzNhbi9XNYFAmGk1doCGwwFCQYGewAACgkQ
ypzNhbi9XNYaagD/eslWmRnEyp12tlLYKamSFCXHbescDpFIEG4SZ3tSG/gBAJj+
UYCroIIYHatvtVRtCmLXFRvx7eXmrFtzBJV7oXAH
=XpqZ
-----END PGP PUBLIC KEY BLOCK-----
```


10.2.4 DevOps Infrastructure

To learn more about our approach to CI (Continuous Integration) strategy used for this release, please see:

Continuous Integration document

Testing

For more details please refer to:

On-device Testing document

10.2.5 Contributions

If you are a developer eager to know more details about Oniro Project or just an enthusiast with a patch proposal, you are welcome to participate in our Oniro Project ecosystem development. To do so, please sign-up using the process described below:

Contributing to Oniro Project document

10.2.6 License

Project manifest as well as project-specific meta-layers, recipes and software packages are, unless specified otherwise, published under Apache 2.0 license. The whole operating system built by users from the project manifest is an aggregate comprised of many third-party components or component groups, each subject to its own license conditions.

Official project release includes only the project manifest as well as project-specific meta-layers, recipes. Any reference binary image, build cache, and any other build artifacts are distributed as a convenience only and are not part of the release itself.