Armv8-R AArch64 Software Stack

Arm Ltd.

Jan 11, 2023

CONTENTS

1 Introduction **1**
- 1.1 Overview . . . 1
- 1.2 Audience . . . 1
- 1.3 Use Cases Overview . . . 2
- 1.4 High Level Architecture . . . 2
- 1.5 Features Overview . . . 5
- 1.6 Documentation Overview . . . 6
- 1.7 Repository Structure . . . 6
- 1.8 Repository License . . . 7
- 1.9 Issue Reporting . . . 7
- 1.10 Feedback and Support . . . 7
- 1.11 Maintainer(s) . . . 7
- 1.12 Reference . . . 7

2 User Guide **9**
- 2.1 Reproduce . . . 9
- 2.2 Extend . . . 15
- 2.3 Borrow . . . 20

3 Developer Manual **21**
- 3.1 Boot Process . . . 21
- 3.2 Components . . . 22
- 3.3 Yocto Layers . . . 27
- 3.4 Armv8-R AArch64 Extras Layer (meta-armv8r64-extras) . . . 29
- 3.5 Applications . . . 31
- 3.6 Validation . . . 34

4 Codeline Management **37**
- 4.1 Overview . . . 37
- 4.2 Yocto Release Process . . . 37
- 4.3 Armv8-R AArch64 Software Stack Branch and Release Process . . . 38

5 License **41**
- 5.1 SPDX Identifiers . . . 41

6 Changelog & Release Notes **43**
- 6.1 Release 5.0 - InterVM Communication with Linux Containers . . . 43
- 6.2 Release 4.0 - Xen . . . 45
- 6.3 Release 3 - UEFI . . . 46
- 6.4 Release 2 - SMP . . . 47
- 6.5 Release 1 - Single Core . . . 47

CHAPTER

ONE

INTRODUCTION

1.1 Overview

The **Armv8-R AArch64 Software Stack** (referred to as **Software Stack** hereafter) provides a collection of software examples intended to run on the Armv8-R AEM FVP[1], which is an implementation of the Armv8-R AArch64 architecture[2]. Together, they are intended to enable exploration of the general Armv8-R AArch64 architecture as well as a specific implementation - Arm® Cortex®-R82[3].

The software provides example deployments of two classes of operating system; Linux as a Rich OS, and Zephyr as an RTOS (Real-Time Operating System) and either can run as bare-metal or under virtualization provided by Xen as a Type-1 hypervisor.

The Poky Linux distribution provides a platform for hosting and deploying user applications with OCI (Open Container Initiative) compliant containers (via Docker) as well as network connectivity. Additionally, the Xen hypervisor provides a communication path for data transfer between the VM guests.

The Yocto layers which define the project contain all the necessary instructions to fetch and build the source, as well as to download the FVP model and launch the examples.

Fixed Virtual Platforms (FVP) are complete simulations of an Arm system, including processor, memory and peripherals. These are set out in a "programmer's view", which gives you a comprehensive model on which to build and test your software. The Fast Models FVP Reference Guide[4] provides the necessary details. The Armv8-R AEM FVP is a free of charge Armv8-R Fixed Virtual Platform. It supports the latest Armv8-R feature set.

This document describes how to build and run the Armv8-R AArch64 Software Stack on the Armv8-R AEM FVP (AArch64) platform. It also covers how to extend features and configurations based on the Software Stack.

1.2 Audience

This document is intended for software, hardware, and system engineers who are planning to use the Armv8-R AArch64 Software Stack, and for anyone who wants to know more about it.

This document describes how to build an image for Armv8-R AArch64 using a Yocto Project[5] build environment. Basic instructions can be found in the Yocto Project Quick Start[6] document.

In addition to having Yocto related knowledge, the target audience also need to have a certain understanding of the following components:

[1] https://developer.arm.com/tools-and-software/simulation-models/fixed-virtual-platforms/arm-ecosystem-models
[2] https://developer.arm.com/documentation/ddi0600/ac
[3] https://developer.arm.com/Processors/Cortex-R82
[4] https://developer.arm.com/docs/100966/latest
[5] https://www.yoctoproject.org/
[6] https://docs.yoctoproject.org/brief-yoctoprojectqs/index.html

- Arm® architectures[7]
- Bootloader (boot-wrapper-aarch64[8] and U-Boot[9])
- Linux kernel[10]
- Zephyr[11]
- Virtualization technology and Xen[12] hypervisor

1.3 Use Cases Overview

There are 3 sub-stacks supported in the Software Stack for either baremetal solution or virtualization solution separately:

- Baremetal solution (without hardware virtualization support)
  - **Baremetal Linux** stack

    Boot a Linux based Rich OS (via U-Boot with UEFI) to command prompt
  - **Baremetal Zephyr** Stack

    Boot Zephyr RTOS directly and run a sample application
- Virtualization solution
  - **Virtualization** stack

    Boot the Xen hypervisor (via U-Boot without UEFI) and start two isolated domains to run Linux Rich OS and Zephyr RTOS in parallel on the fixed number of cores statically allocated by the Xen hypervisor.

    In this stack, Xen provides static shared memory and static event channel support so that the Linux and Zephyr domains running on it can communicate with each other using the RPMsg (Remote Processor Messaging) protocol implemented in the OpenAMP (Open Asymmetric Multi-Processing) framework.

    In addition, the Linux domain can also run a container hosted Nginx web server to serve data for external users visiting through the HTTP protocol.

Instructions for achieving these use cases are given in the article *Reproduce*, subject to the relevant assumed technical knowledge as listed at the *Audience* section in this document.

1.4 High Level Architecture

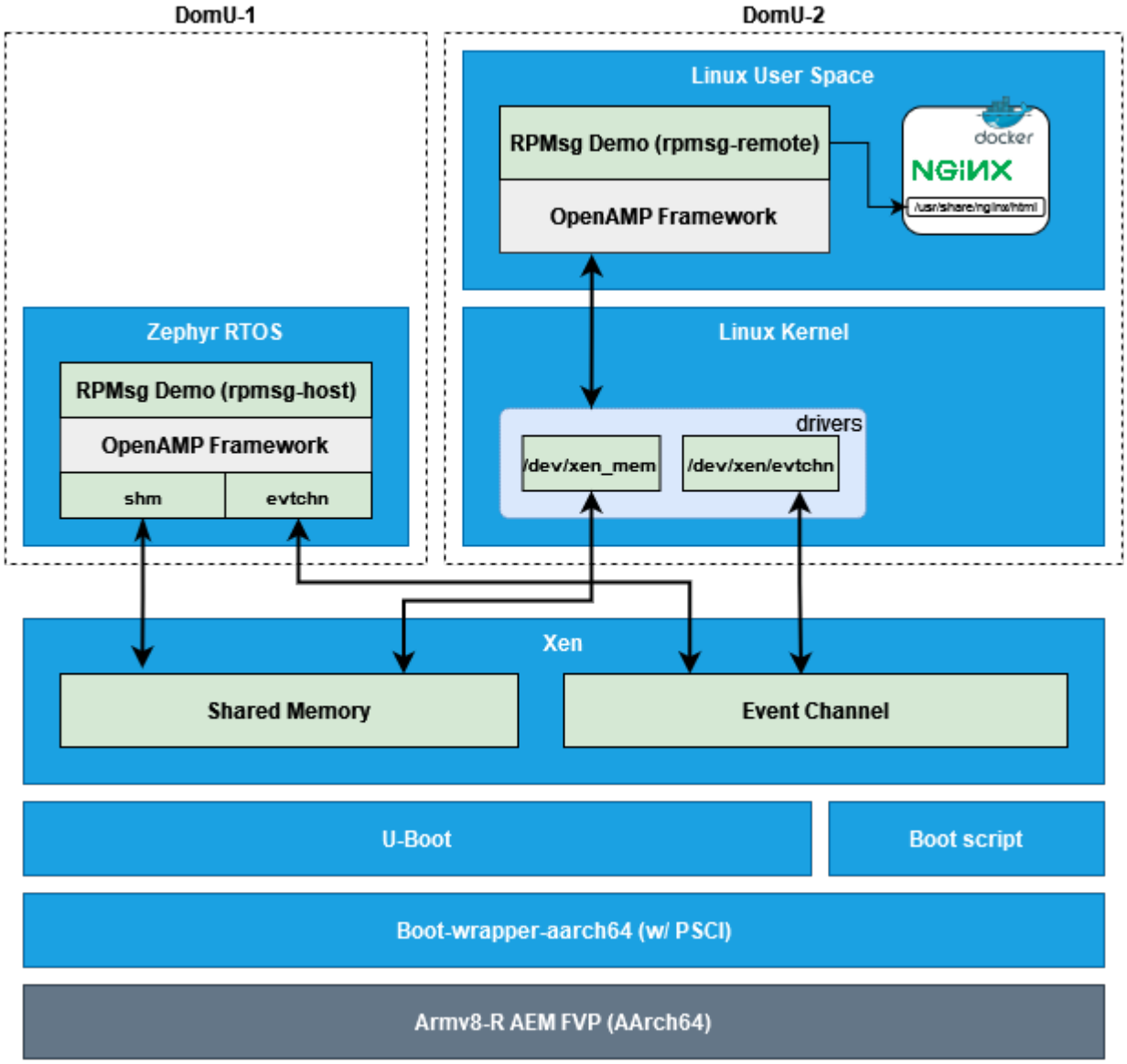
The high-level architecture corresponding to the three use cases (as described in the *Use Cases Overview* section above) supported by this Software Stack is as follows:

The diagram below gives an overview of the components and layers in the implementation of Virtualization stack with Xen hypervisor, adding the main modules required to implement the following two scenarios:

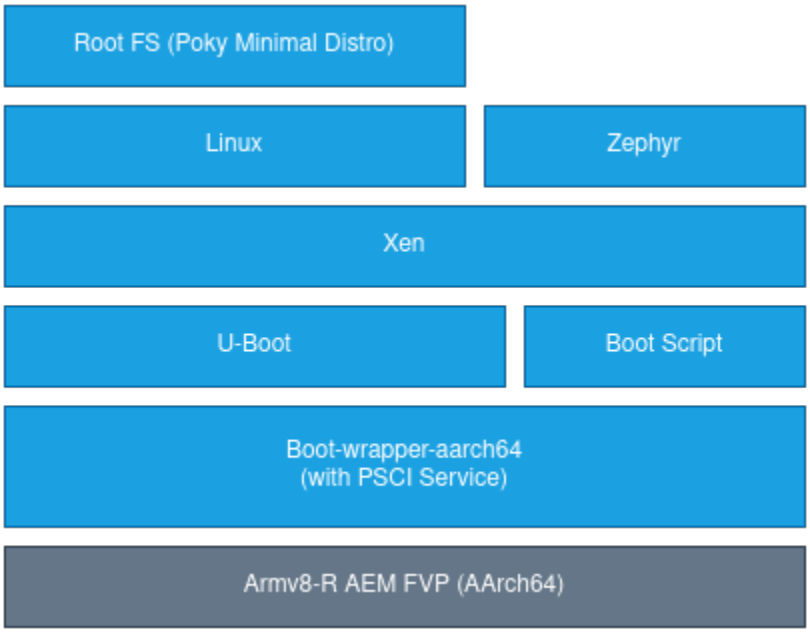
- Inter-VM communication with each other between the two domains running on the Xen hypervisor
- Exposing data with a docker container hosted Nginx web server

[7] https://developer.arm.com/architectures
[8] https://git.kernel.org/pub/scm/linux/kernel/git/mark/boot-wrapper-aarch64.git/
[9] https://www.denx.de/wiki/U-Boot
[10] https://www.kernel.org/
[11] https://www.zephyrproject.org/
[12] https://xenproject.org/

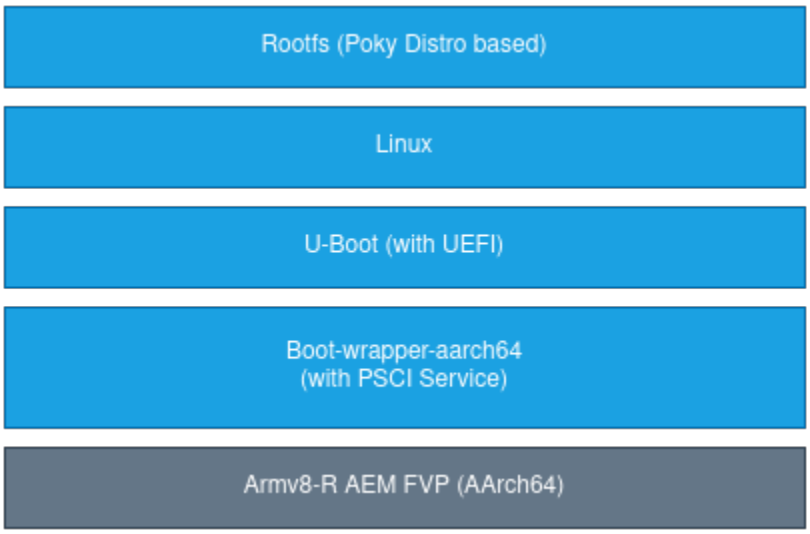
Inter-VM communication is implemented using the OpenAMP framework in the application layer, and supported by the static shared memory and static event channel mechanism provided by Xen in the backend.



The diagram below gives an overview of the components and layers in the implementation of Virtualization stack.



The diagram below gives an overview of the components and layers in the Baremetal Linux stack.



The diagram below gives an overview of the components and layers in the Baremetal Zephyr stack.

1.5 Features Overview

This Software Stack implements the following major features at system level:

- A Zephyr RTOS running on baremetal supporting SMP (Symmetric Multi-Processing)
- A standard Linux Rich OS running on baremetal supporting the following functions:
  - Booting Linux kernel using U-Boot UEFI (Unified Extensible Firmware Interface) and Grub2
  - SMP (Symmetric Multi-Processing)
  - Test suite that can validate the basic functionalities
- A dual-OS system - Zephyr as the RTOS and Linux as the Rich OS - running in parallel using the Xen hypervisor, with supporting the following functions:
  - Booting dom0less Xen from U-Boot
  - SMP in Xen hypervisor and the OSes (both Linux and Zephyr) running in Xen domains
  - Inter-VM communication using RPMsg protocol provided by the OpenAMP framework together with the static shared memory and static event channel provided by Xen
  - Docker in the Linux domain, showing standard application deployments via containers
  - A docker container hosted Nginx web server to serve data for external users visiting through the HTTP protocol
  - Test suite that can validate the functionalities in Zephyr and Linux, as well as the communication between the two OSes and docker running in the Linux domain
  - Classic sample applications in Zephyr

Some other features at component level include:

- Device tree overlay to apply changes on the fly (provided by U-Boot)
- Device pass through support for virtio-net, virtio-blk, virtio-rng, virtio-9p and virtio-rtc (provided by Xen to the Linux kernel)
- PSCI service (provided by boot-wrapper-aarch64)
- Boot from S-EL2 (provided by boot-wrapper-aarch64, U-Boot and Xen)
- Boot-wrapper-aarch64 (as an alternative solution to Trusted-Firmware) and hypervisor (Xen) co-existence in S-EL2 at runtime
- Static shared memory in Xen hypervisor allowing users to statically set up shared memory on a dom0less system, enabling domains to do shm-based communication
- Static event channel established in Xen hypervisor for event notification between two domains

1.6 Documentation Overview

The documentation is structured as follows:

- *User Guide*

  Provides the guide to setup environment, build and run the Software Stack on the target platform, run the provided test suite and validate supported functionalities. Also includes the guidance for how to use extra features, customize configurations, and reuse the components in this Software Stack.

- *Developer Manual*

  Provides more advanced developer-focused details about this Software Stack, include its implementation, and dependencies, etc.

- *Codeline Management*

  Describes the Armv8-R AArch64 Software Stack's branches and release process.

- *License*

  Defines the license under which this Software Stack is provided.

- *Changelog & Release Notes*

  Documents new features, bug fixes, known issues and limitations, and any other changes provided under each release.

1.7 Repository Structure

The Armv8-R AArch64 Software Stack is provided through the v8r64 repository[13], which is structured as follows:

- `meta-armv8r64-extras`

  Yocto layer with the target platform (fvp-baser-aemv8r64[14]) extensions for Armv8-R AArch64. This layer extends the meta-arm-bsp[15] layer in the meta-arm[16] repository. It aims to release the updated and new Yocto Project recipes and machine configurations for Armv8-R AArch64 that are not available in the existing meta-arm and meta-arm-bsp layers in the Yocto Project. This layer also provides image recipes that include all the components needed for a system image to boot, making it easier for the user. Components can be built individually or through an image recipe, which pulls all the components required in an image into one build process.

- `documentation`

  Directory which contains the documentation sources, defined in reStructuredText (.rst) format for rendering via Sphinx[17].

- `config`

  Directory which contains configuration files for running tools on the v8r64 repository, such as files for running automated quality-assurance checks.

- `components`

  Directory containing source code for components which can either be used directly or as part of the Yocto BSP layer or the Yocto meta-armv8r64-extras layer.

[13] https://gitlab.arm.com/automotive-and-industrial/v8r64
[14] https://git.yoctoproject.org/meta-arm/tree/meta-arm-bsp/documentation/fvp-baser-aemv8r64.md
[15] https://git.yoctoproject.org/meta-arm/tree/meta-arm-bsp?h=langdale
[16] https://git.yoctoproject.org/meta-arm/
[17] https://www.sphinx-doc.org/

The Yocto layers which are provided by meta-armv8r64-extras are detailed with the layer structure and dependencies in *Yocto Layers*.

1.8 Repository License

The repository's standard license is the MIT license (more details in *License*), under which most of the repository's content is provided. Exceptions to this standard license relate to files that represent modifications to externally licensed works (for example, patch files). These files may therefore be included in the repository under alternative licenses in order to be compliant with the licensing requirements of the associated external works.

Contributions to the project should follow the same licensing arrangement.

1.9 Issue Reporting

To report issues with the repository such as potential bugs, security concerns, or feature requests, please submit an Issue via GitLab Issues, following the project's Issue template.

1.10 Feedback and Support

To request support please contact Arm at support@arm.com. Arm licensees may also contact with Arm via their partner managers.

1.11 Maintainer(s)

- Robbie Cao <robbie.cao@arm.com>

1.12 Reference

CHAPTER

TWO

USER GUIDE

2.1 Reproduce

This document provides the instructions for setting up the build environment, checking out code, building, running and validating the key use cases.

2.1.1 Environment Setup

The following instructions have been tested on hosts running Ubuntu 20.04. Install the required packages for the build host: https://docs.yoctoproject.org/4.1.1/singleindex.html#required-packages-for-the-build-host

Kas is a setup tool for bitbake based projects. The minimal supported version is `3.0.2` (version `3.1` was used during the development), install it like so:

```
pip3 install --user --upgrade kas==3.1
```

For more details on kas, see https://kas.readthedocs.io/.

To build the recipe for the FVP_Base_AEMv8R model itself, you also need to accept the EULA[1] by setting the following environment variable:

```
export FVP_BASE_R_ARM_EULA_ACCEPT="True"
```

To fetch and build the ongoing development of this Software Stack, follow the instructions in the below sections of this document.

Note: The host machine should have at least 85 GBytes of free disk space for the next steps to work correctly.

2.1.2 Download

Fetch the v8r64 repository into a build directory:

```
mkdir -p ~/fvp-baser-aemv8r64-build
cd ~/fvp-baser-aemv8r64-build
git clone https://gitlab.arm.com/automotive-and-industrial/v8r64 -b v5.0
```

[1] https://developer.arm.com/downloads/-/arm-ecosystem-fvps/eula

2.1.3 Build and Run

The Software Stack supports building and running three sub-stacks: Baremetal Zephyr, Baremetal Linux and Virtualization, which are associated with the use cases described in *Use Cases Overview*. Instructions to build and run these three sub-stacks are as below:

Baremetal Zephyr

Build and run with the below commands:

```
cd ~/fvp-baser-aemv8r64-build

# Build
kas build v8r64/meta-armv8r64-extras/kas/baremetal-zephyr.yml \
    --target zephyr-synchronization

# Output images will be located at
# build/tmp_baremetal-zephyr/deploy/images/fvp-baser-aemv8r64/

# Run
kas shell -k v8r64/meta-armv8r64-extras/kas/baremetal-zephyr.yml \
    -c "../layers/meta-arm/scripts/runfvp --verbose --console"
```

To finish the FVP emulation, you need to close the `telnet` session:

1. Escape to telnet console with `ctrl+]`.
2. Run `quit` to close the session.

A different sample application can be selected by changing the Kas `--target` flag value, e.g.:

```
kas build v8r64/meta-armv8r64-extras/kas/baremetal-zephyr.yml \
    --target zephyr-helloworld
```

There are 3 supported targets `zephyr-helloworld`, `zephyr-synchronization` and `zephyr-philosophers`. See the *Zephyr* section for more information.

Baremetal Linux

Build and run with the below commands:

```
cd ~/fvp-baser-aemv8r64-build

# Build
kas build v8r64/meta-armv8r64-extras/kas/baremetal-linux.yml

# Output images will be located at
# build/tmp_baremetal-linux/deploy/images/fvp-baser-aemv8r64/

# Run
kas shell -k v8r64/meta-armv8r64-extras/kas/baremetal-linux.yml \
    -c "../layers/meta-arm/scripts/runfvp --verbose --console"
```

Virtualization

Build and run with the below commands:

```
cd ~/fvp-baser-aemv8r64-build

# Build
kas build v8r64/meta-armv8r64-extras/kas/virtualization.yml

# Output images are located at
# build/tmp_virtualization/deploy/images/fvp-baser-aemv8r64/

# Run
kas shell -k v8r64/meta-armv8r64-extras/kas/virtualization.yml \
    -c "../layers/meta-arm/scripts/runfvp --verbose"

# Check Xen output.
telnet localhost 5000

# Check guest OS in another terminals after Xen started
# Zephyr
# The zephyr-rpmsg-demo application starts automatically.
telnet localhost 5001

# Linux
# The rpmsg-demo application starts automatically.
telnet localhost 5002
```

Note: When `--console` is specified, port `5002` (which exposes the Linux terminal) is automatically connected through `telnet`. In this case, please do not connect to port 5002 again, as another connection to port `5002` terminates the FVP. If no Linux log is printed, connect to port `5000` (which exposes the Xen terminal) in another terminal. More details of `runfvp` can be found at runfvp.md.

Note: When running the `runfvp` command with the `--verbose` option enabled, you will see the following output:

```
terminal_0: Listening for serial connection on port 5000
terminal_1: Listening for serial connection on port 5001
terminal_2: Listening for serial connection on port 5002
terminal_3: Listening for serial connection on port 5003
```

Among them, port `5000` is assigned to the Xen hypervisor, `5001` to the Zephyr domain, `5002` to the Linux domain, and `5003` unused. If these ports are already occupied (for example, there is already a `runfvp` instance running), then the port number will automatically increase by 4, that is, ports `5004` ~ `5007` will be assigned. In this case, checking the output from the Zephyr and Linux domains requires using ports `5005` and `5006`:

```
telnet localhost 5005
telnet localhost 5006
```

If any port(s) in `5000` ~ `5003` is (are) used by other programs, the FVP will try to find 4 available ports starting from `5000`. Be sure to check the log of `runfvp` (with `--verbose`) to determine the correct port numbers, then apply the determined port number to the `telnet` command.

By default, the `rpmsg-demo` application and the Nginx web server start automatically. `rpmsg-demo` sends status data from the Zephyr domain to the Linux domain, and saves the data to `zephyr-status.html`. The files can be visited through Nginx web server using the following commands in the Linux domain.

```
# The Nginx docker container starts automatically in Linux.
# Check that Nginx is working in Linux.
wget http://localhost/index.html

# Check the rpmsg-demo output in Linux. rpmsg-demo updates this file
# periodically.
wget http://localhost/zephyr-status.html
```

These pages can also be visited from a web browser in the host that is running FVP. By default port `80` in FVP is mapped to port `8080` on host in the Virtualization stack. To browse these pages in the host web browser, use http://localhost:8080/index.html and http://localhost:8080/zephyr-status.html respectively.

More details of *Demo Applications* can be found in *Applications* section.

2.1.4 Validate

The Software Stack contains test suite which are enabled using the environment variable `TESTIMAGE_AUTO=1`. They can be included into the target build to automatically validate the functionality of each stack. The test cases use the Yocto image testing feature[2] to boot the FVP and validate the output.

Baremetal Zephyr

For the Zephyr RTOS, tests are provided to validate the three sample applications supported in this Software Stack. Use the following command to build and run the tests:

```
TESTIMAGE_AUTO=1 kas build v8r64/meta-armv8r64-extras/kas/baremetal-zephyr.yml
```

An example of the test suite output is as follows:

```
Creating terminal default on terminal_0
Skipping - not zephyr-helloworld
Skipping - not zephyr-philosophers
RESULTS:
RESULTS - zephyr_v8r64.ZephyrTests.test_synchronization: PASSED (3.79s)
RESULTS - zephyr_v8r64.ZephyrTests.test_helloworld: SKIPPED (0.00s)
RESULTS - zephyr_v8r64.ZephyrTests.test_philosophers: SKIPPED (0.00s)
SUMMARY:
zephyr-synchronization () - Ran 3 tests in 3.793s
zephyr-synchronization - OK - All required tests passed (successes=1, skipped=2,␣
↪failures=0, errors=0)
Creating terminal default on terminal_0
Skipping - not zephyr-helloworld
Skipping - not zephyr-synchronization
RESULTS:
RESULTS - zephyr_v8r64.ZephyrTests.test_philosophers: PASSED (0.64s)
RESULTS - zephyr_v8r64.ZephyrTests.test_helloworld: SKIPPED (0.00s)
RESULTS - zephyr_v8r64.ZephyrTests.test_synchronization: SKIPPED (0.00s)
```

(continues on next page)

[2] https://docs.yoctoproject.org/test-manual/intro.html

(continued from previous page)

```
SUMMARY:
zephyr-philosophers () - Ran 3 tests in 0.638s
zephyr-philosophers - OK - All required tests passed (successes=1, skipped=2, failures=0,
↪ errors=0)
Creating terminal default on terminal_0
Skipping - not zephyr-philosophers
Skipping - not zephyr-synchronization
RESULTS:
RESULTS - zephyr_v8r64.ZephyrTests.test_helloworld: PASSED (0.73s)
RESULTS - zephyr_v8r64.ZephyrTests.test_philosophers: SKIPPED (0.00s)
RESULTS - zephyr_v8r64.ZephyrTests.test_synchronization: SKIPPED (0.00s)
SUMMARY:
zephyr-helloworld () - Ran 3 tests in 0.731s
zephyr-helloworld - OK - All required tests passed (successes=1, skipped=2, failures=0,␣
↪errors=0)
```

The logs can be viewed at (using `zephyr-synchronization` as an example):

- Testimage output: `build/tmp_baremetal-zephyr/work/armv8r-poky-linux/zephyr-synchronization/3.2.0+gitAUTOINC+4256cd41df-r0/temp/log.do_testimage`
- Console output: `build/tmp_baremetal-zephyr/work/armv8r-poky-linux/zephyr-synchronization/3.2.0+gitAUTOINC+4256cd41df-r0/testimage/default_log`

Baremetal Linux

For the Linux OS, to build the image with the test suite and run it, use the following command:

```
TESTIMAGE_AUTO=1 kas build v8r64/meta-armv8r64-extras/kas/baremetal-linux.yml
```

An example of the test suite output is as follows:

```
Creating terminal default on terminal_0
default: Waiting for login prompt
RESULTS:
RESULTS - linuxboot.LinuxBootTest.test_linux_boot: PASSED (89.56s)
RESULTS - basictests.BasicTests.test_basic_tests: PASSED (155.99s)
RESULTS - sysinfo.SysInfo.test_hostname: PASSED (3.41s)
RESULTS - sysinfo.SysInfo.test_kernel_version: PASSED (3.34s)
SUMMARY:
core-image-minimal () - Ran 4 tests in 252.298s
core-image-minimal - OK - All required tests passed (successes=4, skipped=0, failures=0,␣
↪errors=0)
```

The logs can be viewed at:

- Testimage output: `build/tmp_baremetal-linux/work/fvp_baser_aemv8r64-poky-linux/core-image-minimal/1.0-r0/temp/log.do_testimage`
- Console output: `build/tmp_baremetal-linux/work/fvp_baser_aemv8r64-poky-linux/core-image-minimal/1.0-r0/testimage/default_log`

Virtualization

For the Virtualization stack, to build the image with the test suite and run it, use the following command:

```
XEN_DOM0LESS_DOM_LINUX_DEMO_AUTORUN=0 \
TESTIMAGE_AUTO=1 \
    kas build v8r64/meta-armv8r64-extras/kas/virtualization.yml
```

This runs the same test cases as for the baremetal Linux and Zephyr stacks, as well as an additional test case against the Xen console output.

Note: When setting `TESTIMAGE_AUTO=1` for automatic testing, `XEN_DOM0LESS_DOM_LINUX_DEMO_AUTORUN` must be set to 0, and vice versa. That is, `TESTIMAGE_AUTO` and `XEN_DOM0LESS_DOM_LINUX_DEMO_AUTORUN` cannot be set to 1 at the same time, because demo application autorun and `testimage` both start the `rpmsg-remote` application, which will cause `testimage` to fail.

An example of the test suite output is as follows:

```
Creating terminal default on terminal_2
Creating terminal xen on terminal_0
Creating terminal zephyr_v8r64 on terminal_1
default: Waiting for login prompt
ptest-runner started
Skipping - not zephyr-helloworld
Skipping - not zephyr-philosophers
Skipping - not zephyr-synchronization
RESULTS:
RESULTS - linuxboot.LinuxBootTest.test_linux_boot: PASSED (145.80s)
RESULTS - basictests.BasicTests.test_basic_tests: PASSED (181.35s)
RESULTS - linuxlogin.LinuxLoginTest.test_linux_login: PASSED (4.58s)
RESULTS - ptest_docker.PtestDockerTests.test_ptestdocker: PASSED (439.05s)
RESULTS - rpmsg.RpmsgTests.test_rpmsg_demo: PASSED (62.72s)
RESULTS - rpmsg.RpmsgTests.test_zephyr_boot: PASSED (0.00s)
RESULTS - sysinfo.SysInfo.test_hostname: PASSED (5.30s)
RESULTS - sysinfo.SysInfo.test_kernel_version: PASSED (3.28s)
RESULTS - xen.XenBoot.test_boot: PASSED (0.00s)
RESULTS - zephyr_v8r64.ZephyrTests.test_helloworld: SKIPPED (0.00s)
RESULTS - zephyr_v8r64.ZephyrTests.test_philosophers: SKIPPED (0.00s)
RESULTS - zephyr_v8r64.ZephyrTests.test_synchronization: SKIPPED (0.00s)
SUMMARY:
core-image-minimal () - Ran 12 tests in 842.089s
core-image-minimal - OK - All required tests passed (successes=9, skipped=3, failures=0,␣
↪errors=0)
```

The logs can be viewed at:

- Testimage output: `build/tmp_virtualization/work/fvp_baser_aemv8r64-poky-linux/core-image-minimal/1.0-r0/temp/log.do_testimage`
- Xen console output: `build/tmp_virtualization/work/fvp_baser_aemv8r64-poky-linux/core-image-minimal/1.0-r0/testimage/xen_log`
- Zephyr console output: `build/tmp_virtualization/work/fvp_baser_aemv8r64-poky-linux/core-image-minimal/1.0-r0/testimage/zephyr_v8r64_log`

- Linux console output: `build/tmp_virtualization/work/fvp_baser_aemv8r64-poky-linux/core-image-minimal/1.0-r0/testimage/default_log`

2.1.5 Reference

Instructions for setting up the build environment, checking out code, building, running and validating the key use cases.

2.2 Extend

This document provides the guides showing how to use extra features, customize configurations in this Software Stack.

2.2.1 Extra Features

Networking

The FVP is configured by default to use "user mode networking", which simulates an IP router and DHCP server to avoid additional host dependencies and networking configuration. Outbound connections work automatically, e.g. by running:

```
wget www.arm.com
```

Inbound connections require an explicit port mapping from the host. By default, port `8022` on the host is mapped to port `22` on the FVP, so that the following command will connect to an ssh server running on the FVP:

```
ssh root@localhost -p 8022
```

To map other ports from host, add the parameter containing the port mapping in the command as below:

```
kas shell -k \
    v8r64/meta-armv8r64-extras/kas/virtualization.yml \
    -c "../layers/meta-arm/scripts/runfvp \
        --verbose --console -- --parameter \
        'bp.virtio_net.hostbridge.userNetPorts=8022=22,8080=80,5555=5555'"
```

Note: User mode networking does not support ICMP, so `ping` will not work.

More details on this topic can be found at User mode networking[1].

[1] https://developer.arm.com/documentation/100964/1118/Introduction-to-Fast-Models/User-mode-networking?lang=en

File Sharing between Host and FVP

It is possible to share a directory between the host machine and the FVP using the virtio P9 device component included in the kernel. To do so, create a directory to be mounted from the host machine:

```
mkdir /path/to/host-mount-dir
```

Then, add the following parameter containing the path to the directory when launching the model:

```
--parameter 'bp.virtiop9device.root_path=/path/to/host-mount-dir'
```

e.g. for the virtualization build:

```
kas shell -k \
    v8r64/meta-armv8r64-extras/kas/virtualization.yml \
    -c "../layers/meta-arm/scripts/runfvp \
        --verbose --console -- --parameter \
        'bp.virtiop9device.root_path=/path/to/host-mount-dir'"
```

Once you are logged into the FVP, the host directory can be mounted in a directory on the model using the following command:

```
mount -t 9p -o trans=virtio,version=9p2000.L FM /path/to/fvp-mount-dir
```

2.2.2 Customize Configuration

Customizing the Zephyr Configuration

The Zephyr repository contains two relevant board definitions, `fvp_baser_aemv8r` and `fvp_baser_aemv8r_smp`, each of which provides a base defconfig and device tree for the build. In the Yocto build, the board definition is selected dynamically based on the number of CPUs required by the application recipe.

The defconfig can be extended by adding one or more `.conf` files to `SRC_URI` (which are passed to the `OVERLAY_CONFIG` Zephyr configuration flag).

Note: The file extension for Zephyr config overlays (`.conf`) is different to the extension used by config fragments in other recipes (`.cfg`), despite the similar functionality.

The device tree can be modified by adding one or more `.overlay` files to `SRC_URI` (which are passed to the `DTC_OVERLAY_FILE` Zephyr configuration flag). These overlays can modify, add or remove nodes in the board's device tree.

For an example of these overlay files and how to apply them to the build, see the modifications to run Zephyr applications on Xen in directory meta-armv8r64-extras/dynamic-layers/virtualization-layer/recipes-kernel/zephyr-kernel/

The link Important Build System Variables[2] provides more information.

Note: Zephyr's device tree overlays have a different syntax from U-Boot's device tree overlays.

---

[2] https://docs.zephyrproject.org/3.2.0/develop/application/index.html#important-build-system-variables

Customizing the Xen Domains

The default configuration contains two pre-configured domains: `XEN_DOM0LESS_DOM_LINUX` and `XEN_DOM0LESS_DOM_ZEPHYR`, with the BitBake varflags set appropriately. These varflags define where to find the domain binaries in the build configuration, where to load them in memory at runtime and how to boot the domain. Note that no attempts are made to validate overlapping memory regions, or even whether the defined addresses fit in the FVP RAM. For information, see the notes in the file meta-armv8r64-extras/classes/xen_dom0less_config.bbclass.

Currently, the default images that these two pre-configured domains run on are: one Linux rootfs image (`core-image-minimal`) for `XEN_DOM0LESS_DOM_LINUX` and one Zephyr application (`zephyr-rpmsg-demo`) for `XEN_DOM0LESS_DOM_ZEPHYR`.

The default Linux rootfs image can be configured using the variable `XEN_DOM0LESS_LINUX_IMAGE` and the default Zephyr application can be configured using the variable `XEN_DOM0LESS_ZEPHYR_APPLICATION`. For example, to use `core-image-base` and `zephyr-helloworld` instead of the defaults, run:

```
XEN_DOM0LESS_LINUX_IMAGE="core-image-base" \
XEN_DOM0LESS_ZEPHYR_APPLICATION="zephyr-helloworld" \
    kas build v8r64/meta-armv8r64-extras/kas/virtualization.yml
```

2.2.3 Customize Parameter

Customizing the FVP Parameters

FVPs can be configured some aspects of their behavior through command-line parameters. When starting FVP using the `runfvp` script, the parameters can be customized by putting them after the separator `--`. For example, in the following command, port mapping is customized by the additional FVP parameter `bp.virtio_net.hostbridge.userNetPorts`, which will override the default value of the same parameter defined in meta-armv8r64-extras/classes/xen_dom0less_image.bbclass.

```
kas shell -k \
    v8r64/meta-armv8r64-extras/kas/virtualization.yml \
    -c "../layers/meta-arm/scripts/runfvp \
        --verbose --console -- --parameter \
        'bp.virtio_net.hostbridge.userNetPorts=8022=22,8080=80,5555=5555'"
```

The default parameters for the Software Stack to run FVP are defined in the following files:

- https://git.yoctoproject.org/meta-arm/tree/meta-arm-bsp/conf/machine/fvp-baser-aemv8r64.conf?h=langdale
- meta-armv8r64-extras/classes/zephyr-fvpboot.bbclass
- meta-armv8r64-extras/classes/xen_dom0less_image.bbclass

Among them, fvp-baser-aemv8r64.conf defines the base FVP parameters of the Software Stack, and is also the default FVP parameter of the Baremetal Linux stack. Based on the above default parameters, meta-armv8r64-extras/classes/zephyr-fvpboot.bbclass customizes the FVP parameters for Baremetal Zephyr stack, and meta-armv8r64-extras/classes/xen_dom0less_image.bbclass customizes the FVP parameters for the Virtualization stack.

You can run the command below to check the description and optional values of the FVP parameters. For more details about FVP and its supported parameters, see Fast Models FVP Reference Guide.

```
kas shell -k v8r64/meta-armv8r64-extras/kas/virtualization.yml \
    -c "../layers/meta-arm/scripts/runfvp -- --list-params"
```

Warning: The default parameters are elaborately tuned, and changing parameters you don't understand may lead to unpredictable results.

Customizing Build Environment Parameters

The Software Stack can be configured some aspects of its target image and runtime behavior at build time via command-line environment variables. The build environment variables and parameters supported by the Software Stack and their scope of application are detailed below.

- `TESTIMAGE_AUTO`

  Controls if `testimage` runs automatically after an image build.

  Options:

  - `0` (default)

    Enable to run test suite automatically after an image build

  - `1`

    Disable to run test suite automatically after an image build

  Applicable:

  - **Baremetal Zephyr**
  - **Baremetal Linux**
  - **Virtualization**

  See the section *Validation* for more details.

- `XEN_DOM0LESS_ZEPHYR_APPLICATION`

  Specifies the Zephyr application recipe to be used.

  Options:

  - `zephyr-rpmsg-demo` (default)
  - `zephyr-helloworld`
  - `zephyr-synchronization`
  - `zephyr-philosophers`

  Applicable:

  - **Baremetal Zephyr**
  - **Virtualization**

  There is one exception is that `zephyr-rpmsg-demo` only works on the Virtualization stack. See the *Zephyr Sample Applications* section for more information.

- `XEN_DOM0LESS_DOMAINS`

  Specifies which domain(s) will be enabled on Xen.

  Options:

  - `"XEN_DOM0LESS_DOM_LINUX XEN_DOM0LESS_DOM_ZEPHYR"` (default)
  - `"XEN_DOM0LESS_DOM_LINUX"`

  - "XEN_DOM0LESS_DOM_ZEPHYR"

  Applicable:

  - **Virtualization**

  See the section *Customizing the Xen Domains* for more details.

- XEN_DOM0LESS_LINUX_IMAGE

  Configures the Linux rootfs image.

  Options:

  - core-image-minimal (default)

  Applicable:

  - **Virtualization**

  The Images section of the Yocto Manual explains details about Linux rootfs images. It also lists more images that may be used as XEN_DOM0LESS_LINUX_IMAGE but are not officially supported by this Software Stack.

- XEN_DOM0LESS_DOM_LINUX_DEMO_AUTORUN

  Controls if the demo application runs automatically after Linux boot.

  Options:

  - 0

    Disable to run *Demo Applications* automatically in the Linux domain after system boot

  - 1 (default)

    Enable to run *Demo Applications* automatically in the Linux domain after system boot

  Applicable:

  - **Virtualization**

An example to build zephyr-synchronization and run testimage to validate after image build on Baremetal Zephyr stack, use the below command:

```
XEN_DOM0LESS_ZEPHYR_APPLICATION="zephyr-synchronization" \
    kas build v8r64/meta-armv8r64-extras/kas/baremetal-zephyr.yml
```

Another example of combining parameters, to run only Linux domain on Xen using core-image-base as rootfs, and disable testimage at build time and not automatically run demo application at run-time:

```
TESTIMAGE_AUTO=0 \
XEN_DOM0LESS_DOM_LINUX_DEMO_AUTORUN=0 \
XEN_DOM0LESS_DOMAINS="XEN_DOM0LESS_DOM_LINUX" \
XEN_DOM0LESS_LINUX_IMAGE="core-image-base" \
    kas build v8r64/meta-armv8r64-extras/kas/virtualization.yml
```

2.2.4 Reference

Guides showing how to use extra features, customize configurations in this Software Stack.

2.3 Borrow

This document is a deeper explanation of how to reuse the components, patches in this Software Stack.

2.3.1 Reusing the Firmware Patches

The firmware (`linux-system.axf`) consists of U-Boot and the base device tree, bundled together with boot-wrapper-aarch64. Both U-Boot and boot-wrapper-aarch64 are patched to support the Armv8-R AArch64 architecture. These patches live in `meta-arm-bsp`[1].

For further details on the boot-wrapper-aarch64 patches see the *Boot-wrapper-aarch64* section in *Developer Manual*. The U-Boot *Additional Patches* section provides more details on the U-Boot patches.

The base device tree[2] can be found in the meta-arm-bsp Yocto layer.

2.3.2 Reusing the Xen Patches

The patch series for Armv8-R AArch64 with MPU support, which are located in meta-armv8r64-extras/dynamic-layers/virtualization-layer/recipes-extended/xen/files, initialize the PoC (Proof of Concept) of Xen on the Armv8-R AArch64 architecture. These patches are implemented based on Xen 4.17, and the work to upstream these patches to the Xen mainline is in progress.

2.3.3 Reference

A deeper explanation of how to reuse the components, patches in this Software Stack.

[1] https://git.yoctoproject.org/meta-arm/tree/meta-arm-bsp?h=langdale
[2] https://git.yoctoproject.org/meta-arm/tree/meta-arm-bsp/recipes-kernel/linux/files/fvp-baser-aemv8r64/fvp-baser-aemv8r64.dts?h=langdale

CHAPTER

THREE

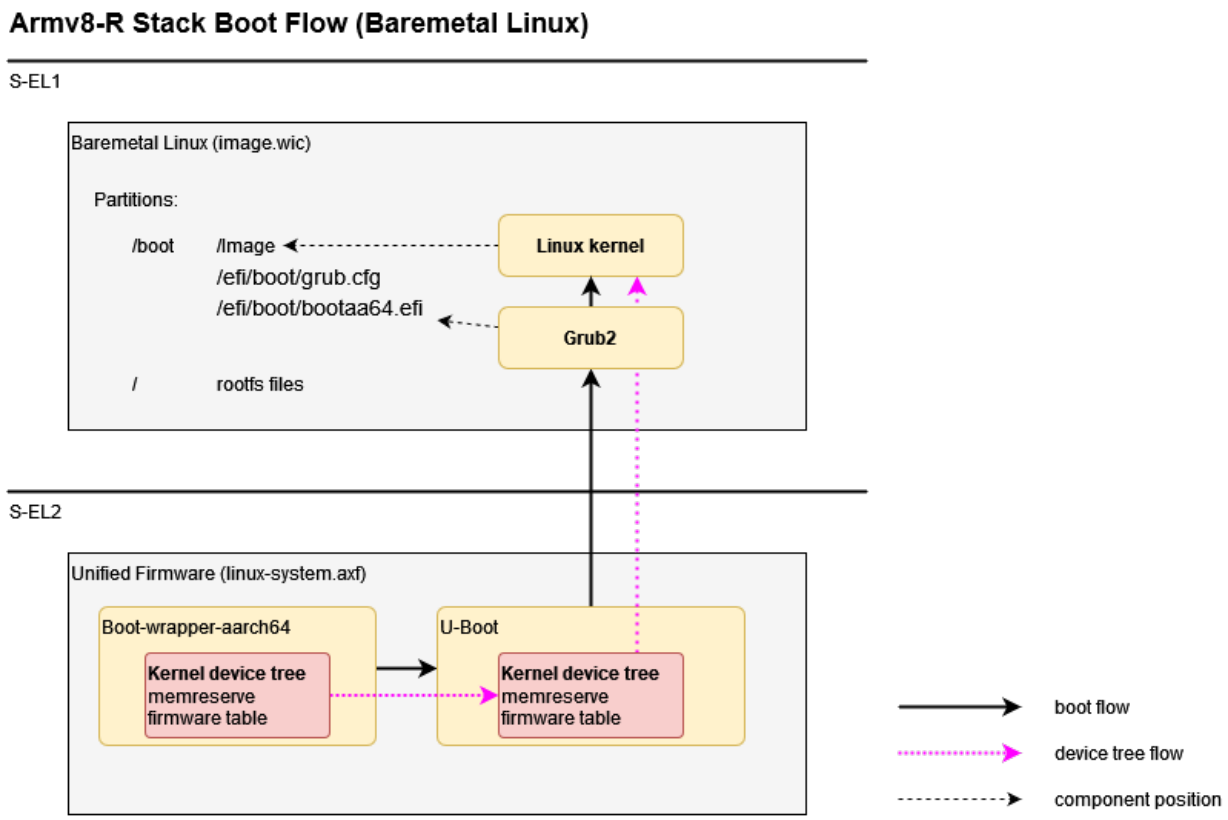
DEVELOPER MANUAL

3.1 Boot Process

As described in the *High Level Architecture* section of the *Introduction*, the system can boot in 3 different ways. The corresponding boot flow is as follows:

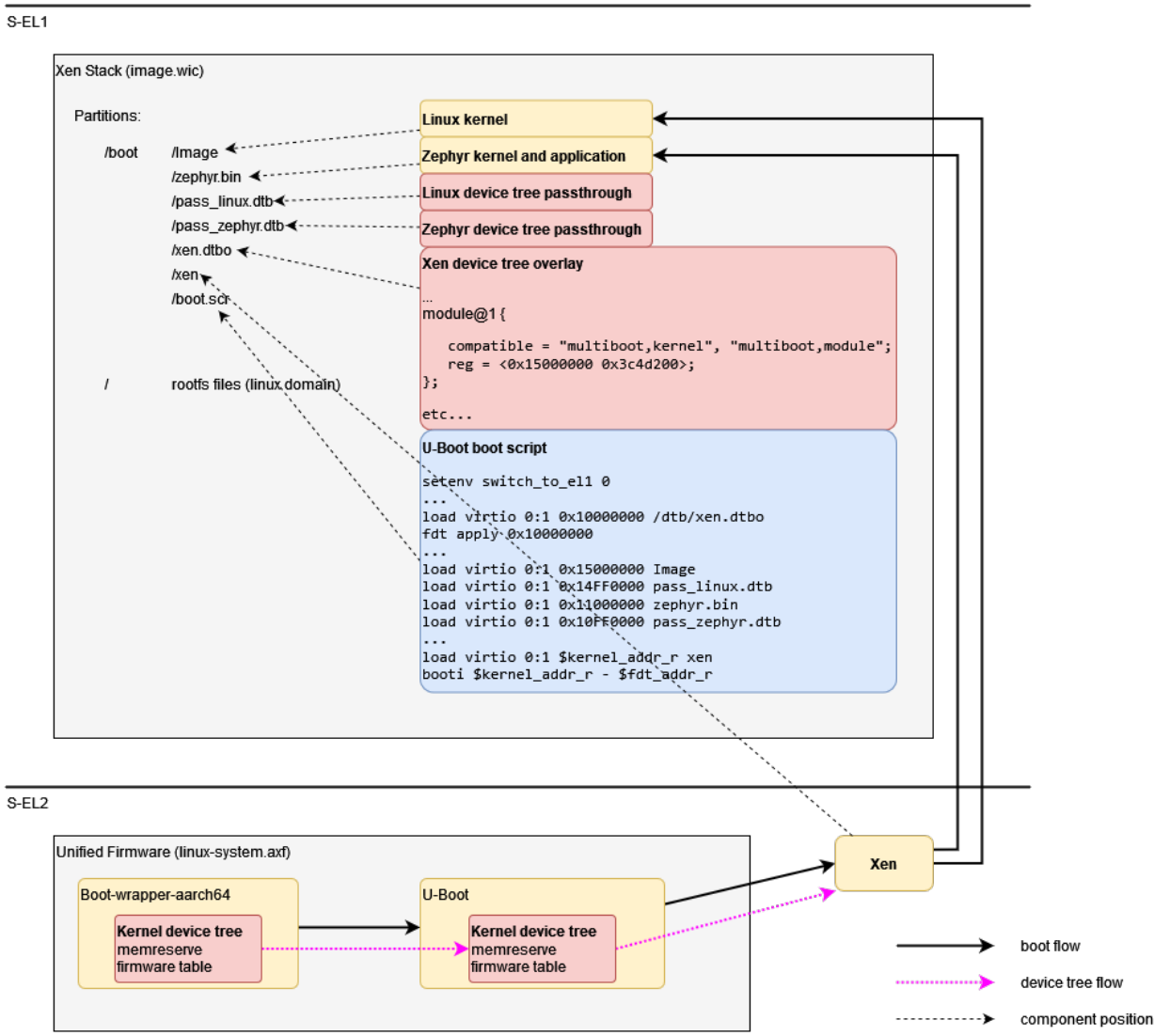
The booting process of Baremetal Zephyr is quite straightforward: Zephyr boots directly from the reset vector after system reset.

For Baremetal Linux, the booting process is as the following diagram:



And the booting process of Virtualization solution is as the following diagram:

Armv8-R Stack Boot flow (Virtualization - Xen Hypervisor)



The *Boot Sequence* section provides more details.

3.2 Components

3.2.1 Boot-wrapper-aarch64

The Armv8-R AArch64 application level programmers' model differs from the Armv8-A AArch64 profile in the following ways:

- Armv8-R AArch64 supports only a single Security state, Secure.
- EL2 is mandatory.
- EL3 is not supported.
- Armv8-R AArch64 supports the A64 ISA instruction set with some modifications.

See the section B1.1.1 of Arm Architecture Reference Manual Supplement for more details.

In this Software Stack, boot-wrapper-aarch64 works as an alternative Trusted-Firmware solution, to solve the difference in the startup process due to the above differences.

The original general boot-wrapper is a fairly simple implementation of a boot loader intended to run under an ARM Fast Model and boot Linux. In this Software Stack, boot-wrapper-aarch64 implements the following functions for the Armv8-R AArch64 architecture:

- S-EL2 booting for Armv8-R AArch64
- Supports to boot Linux or U-Boot from S-EL1
- Supports to boot Xen hypervisor or U-Boot from S-EL2
- Provides PSCI services (`CPU_ON` / `CPU_OFF`) for booting SMP Linux on baremetal
- Provides PSCI services (`CPU_ON` / `CPU_OFF`) to support Xen SMP boot by introducing libfdt to manipulate Flattened Device Trees dynamically and reserve `/memreserve` to prevent from overriding PSCI services

Boot-wrapper-aarch64 is implemented in https://git.kernel.org/pub/scm/linux/kernel/git/mark/boot-wrapper-aarch64.git/, with additional patches applied in the meta-arm-bsp layer.

Two new compilation options have been added in these patches: `--enable-psci` and `--enable-keep-el`.

Flag `--enable-psci` can be used to choose PSCI method between *Secure Monitor Call* (SMC) and *Hypervisor Call* (HVC). To use `smc`, select `--enable-psci` or `--enable-psci=smc`. To use `hvc`, select `--enable-psci=hvc`.

Armv8-R AArch64 does not support `smc`, thus `hvc` is selected in this stack.

Flag `--enable-keep-el` can be used to enable boot-wrapper-aarch64 to boot next stage at S-EL2. As the Armv8-R AArch64 architecture boots from S-EL2, if the next stage requires executing from S-EL2, the boot-wrapper-aarch64 shall not drop to S-EL1.

Configuration of these two options in this stack can be found here.

3.2.2 Bootloader (U-Boot)

The stack's bootloader is implemented by U-Boot (version 2022.07), with additional patches applied in the meta-arm-bsp layer.

Additional Patches

The patches are based on top of U-Boot's "vexpress64" board family (which includes the Juno Versatile Express development board and the `FVP_Base_RevC-2xAEMvA` model) because the `FVP_Base_AEMv8R` model has a similar memory layout. The board is known in U-Boot as `BASER_FVP` and is implemented through additions to the `vexpress_aemv8.h` header file and the `vexpress_aemv8r_defconfig` default configuration file.

As well as supporting the `BASER_FVP` memory map, to allow running Xen at `S-EL2` it is required to run U-Boot at `S-EL2` as well, which has the following implications:

- It is required to initialize the S-EL2 Memory Protection Unit (MPU) to support unaligned memory accesses.
- Boot-wrapper-aarch64's PSCI handler uses the exception vector at S-EL2 (`VBAR_EL2`). By default U-Boot overwrites this for its panic handler.
- We cannot disable hypercalls in `HCR_EL2`, as `HVC` instructions are used for PSCI services.
- A mechanism is required to decide at runtime whether to boot the next stage at S-EL2 (for Xen) or S-EL1 (for Linux).

Additional patches have therefore been added to:

- Configure Memory Protection Units (MPU). Additionally, add logic to detect whether a Memory Management Unit (MMU) is present at the current exception level and if not, trigger the MPU initialization and deinitialization. It is only possible to determine which memory system architecture is active at S-EL1 from S-EL2 (system register `VTCR_EL2`), so at S-EL1 assume the MMU is configured for backwards compatibility.
- Disable setting the exception vectors (`VBAR_EL2`). There is already logic to disable exception vectors for Secondary Program Loader (SPL) builds, so this has been extended to allow disabling the vectors for non-SPL builds too.
- Make disabling hypercalls when switching to S-EL1 configurable.
- Extend the `ARMV8_SWITCH_TO_EL1` functionality:
  - By adding support for switching to S-EL1 for EFI (Extensible Firmware Interface) booting (it was previously only implemented for `booti` and `bootm`).
  - The environment variable `armv8_switch_to_el1` has been added to allow the boot exception level to be configured at runtime, which overrides the compile-time option.
- Disable setting the `COUNTER_FREQUENCY`. The value it's set to might be different from the value used by the first-stage bootloader.

To support amending the device tree at runtime, there is also a patch to enable `CONFIG_LIBFDT_OVERLAY` for the `BASER_FVP`.

Boot Sequence

U-Boot's "distro" feature provides a standard `bootcmd` which automatically attempts a pre-configured list of boot methods. For the `BASER_FVP`, these include:

1. Load a boot script from memory.
2. Load a boot script stored at `/boot.scr` on any supported partition on any attached block device.
3. Load a "removable media" EFI payload stored at `/EFI/boot/bootaa64.efi` on any supported partition on any attached block device.

A boot script is a file (compiled using U-Boot's `mkimage` command) which contains commands that are executed by U-Boot's interpreter.

For baremetal Linux, option 3 above is used. Grub2 (in EFI mode) is installed at the required path in the boot partition of the disk image, which is attached to the FVP virtio block interface. Grub2 is configured with a single menu item to boot the Linux kernel, the image for which is stored in the same partition.

For virtualization, option 2 above is used. A boot script is added to the boot partition of the disk image, which:

- Loads a device tree overlay file from the same partition and applies it to the firmware's device tree.
- Loads the Xen module binaries and passthrough device trees to the required memory addresses.
- Loads the Xen binary itself.
- Sets the `armv8_switch_to_el1` environment variable to `n`, so that Xen will boot at S-EL2.
- Boots Xen using the `booti` boot method.

Option 1 above is not used.

3.2.3 Hypervisor (Xen)

This Software Stack uses Xen as the Type-1 hypervisor for hardware virtualization, which makes it possible to run multiple operating systems (Linux as the Rich OS and Zephyr as RTOS) in parallel on a single Armv8-R AEM FVP model (AArch64 mode). Xen in this Software Stack is implemented by version 4.17 and the additional patches in directory meta-armv8r64-extras/dynamic-layers/virtualization-layer/recipes-extended/xen/files to support the Armv8-R AArch64 architecture.

In addition to the differences mentioned in the *Boot-wrapper-aarch64* section, the Armv8-R AArch64 system level architecture differs from the Armv8-A AArch64 profiles in the following ways:

- Armv8-R AArch64 provides a Protected Memory System Architecture (PMSA) based virtualization model.
- The Armv8-R AArch64 implementation supports PMSA at S-EL1 and S-EL2, based on Memory Protection Unit (MPU).
- Armv8-R AArch64 supports Virtual Memory System Architecture (VMSA), in which provides a Memory Management Unit (MMU), as an optional memory system architecture at S-EL1. In other words: optional S-EL1 MMU is supported in the Armv8-R AArch64 implementation.

These patches are mainly based on the above differences to provide support for Xen on the Armv8-R AArch64 architecture, enabling the virtualization use case described in *Use Cases Overview* in the following ways:

- Enable MPU at S-EL2 to introduce virtualization at S-EL2
- Support MPU at S-EL1 to host a Zephyr RTOS
- Support MMU at S-EL1 to host a Linux Rich OS

And have the following functions to complete the entire virtualization use case:

- The RTOS (Zephyr) domain with S-EL1 MPU and the Rich OS (Linux) domain with S-EL1 MMU can share the same physical cores, to make these two types of OSes can run in parallel (New feature for Xen supported by Armv8-R AArch64 only)
- Xen with S-EL2 MPU isolates the RTOS workload and the Rich OS workload
- Xen shares device tree nodes with boot-wrapper-aarch64 to implement SMP (Symmetric Multi-Processing)
- Xen co-exists with boot-wrapper-aarch64 providing PSCI services at S-EL2
- Xen runs in secure state only, thus OSes in Xen domains run in secure state too

---

Note: The Armv8-R AEM FVP model supports 32 MPU regions by default, which is the typical setting configured by the parameters of `cluster0.num_protection_regions_s1` and `cluster0.num_protection_regions_s2`. Any other settings about the number of MPU regions may cause unexpected malfunctionality or low performance.

In this implementation, Xen utilizes the "dom0less" feature to create 2 DomUs at boot time. Information about the DomUs to be created by Xen is passed to the hypervisor via device tree. The resources for DomUs, including memory, number of vCPUs, supported devices, etc., are all fully static allocated at compile time.

The resources for DomUs also include static shared memory and static event channel. The static shared memory device tree nodes allow users to statically set up shared memory on a dom0less system, enabling domains to do shm-based communication. The static event channel communication can be established statically between two domains (not only between two domUs, also between dom0 and domU). Static event channel connection information between domains will be passed to Xen via the device tree node. The static event channel will be created and established in Xen before the domain started. Domain only needs hypercall `EVTCHNOP_send` to send notifications to the remote guest.

The documentation at this link provides more details about the device tree.

The static shared memory and static event channel are supported by Xen 4.17, as listed in the Xen Project 4.17 Feature List.

3.2.4 Linux Kernel

The Linux kernel in this stack is version 5.19 at https://git.yoctoproject.org/linux-yocto/.

The kernel configuration for Armv8-R AArch64 in this stack is defined in the cfg and scc files in the meta-arm-bsp layer in the meta-arm repository. The device tree can also be found in the meta-arm-bsp layer.

Devices supported in the kernel:

- serial
- virtio 9p
- virtio disk
- virtio network
- virtio rng
- watchdog
- rtc

In addition to the above configurations, Linux running in the Xen domain as a guest OS also enables the following additional configurations to support static shared memory, static event channel, docker, etc.:

- File meta-armv8r64-extras/dynamic-layers/virtualization-layer/recipes-extended/xen/files/fvp-baser-aemv8r64.cfg introduces Xen support for fvp-baser-aemv8r64
- File xen.cfg contains kernel features needed to run as a Xen guest OS
- File docker.cfg contains kernel features needed by docker
- File meta-armv8r64-extras/dynamic-layers/virtualization-layer/recipes-kernel/linux/files/fvp-baser-aemv8r64/kmeta-extra/features/xen-shmem.cfg contains kernel features to support Xen static shared memory
- Kernel features that support Xen static event channels are enabled by default, so no additional configuration is required

Additional Patches

The static shared memory and static event channel drivers are enabled in Linux by additional patches in directory meta-armv8r64-extras/dynamic-layers/virtualization-layer/recipes-kernel/linux/files/fvp-baser-aemv8r64/kmeta-extra/features. These patches provide the following features:

- Enable `xen,dom0less` to the Linux domain.
- Add a new IOCTL type `IOCTL_EVTCHN_BIND` to bind statically allocated static event channel port.
- Add a new `xen,shared-info` property for information sharing between Xen and domains in `xen,dom0less` system.
- Add static shared memory driver.

3.2.5 Zephyr

The Zephyr OS is an open source real-time operating system based on a small-footprint kernel designed for user on resource-constrained and embedded systems.

The stack supports to run Zephyr (version 3.2.0) either on baremetal, or as a Xen domain in the virtualization solution.

Zephyr is implemented in https://github.com/zephyrproject-rtos/zephyr, and supports the Armv8-R AArch64 architecture using Arm FVP BaseR AEMv8-R board listed in Zephyr supported boards.

Zephyr provides many demos and sample programs. This Software Stack supports the following 3 classic samples:

- Hello World
- Synchronization Sample
- Dining Philosophers

A demo application is also provided in this Software Stack to showcase the communication between the Zephyr domain and the Linux domain using OpenAMP RPMsg, based on the static shared memory and static event channel provided by the Xen hypervisor.

Additional Patches

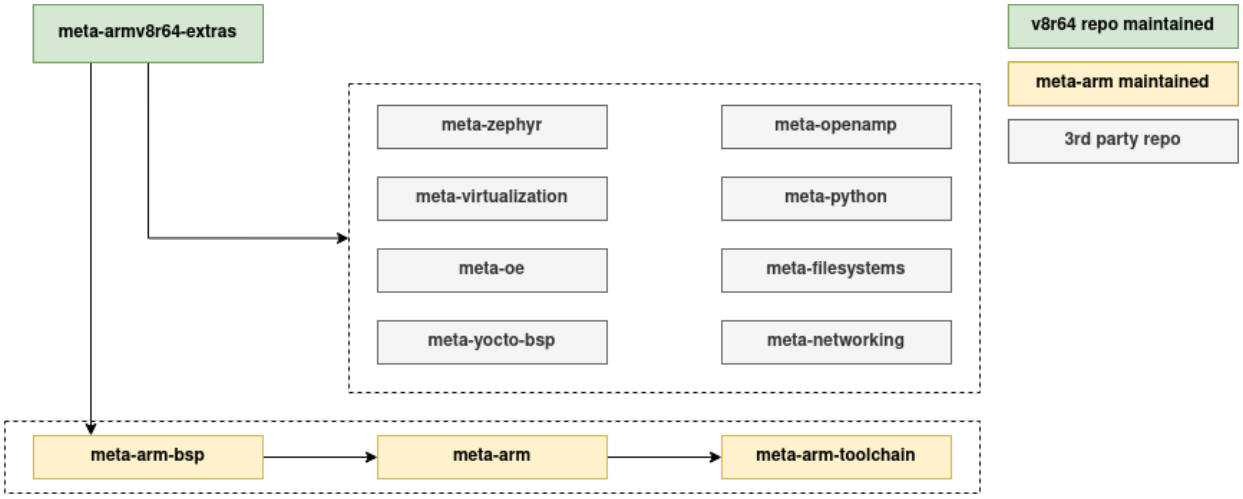
The static shared memory and static event channel drivers are enabled in Zephyr by additional patches in directory meta-armv8r64-extras/dynamic-layers/virtualization-layer/recipes-kernel/zephyr-kernel/files. These patches provide the following features:

- Add hypervisor node in device tree to run Zephyr on Xen.
- Add MPU static shared memory region.
- Add Kconfig item `XEN` and `XEN_DOMAIN_SHARED_MEM` to enable Xen driver and Xen domain static shared memory in Zephyr.
- Save event comes before channel is bound, so that the event won't be lost if the domain which receives events has not bound.

3.3 Yocto Layers

The Armv8-R AArch64 Software Stack is provided through the GitLab repository: https://gitlab.arm.com/automotive-and-industrial/v8r64. The v8r64 repository provides a Yocto Project compatible layer – `meta-armv8r64-extras` – with target platform fvp-baser-aemv8r64 extensions for Armv8-R AArch64. Currently this layer extends the `fvp-baser-aemv8r64` machine definition from the meta-arm-bsp layer in the meta-arm repository.

The following diagram illustrates the layers which are integrated in the Software Stack, which are further expanded below.



- poky layers (meta, meta-poky)
  - URL: https://git.yoctoproject.org/poky/?h=langdale
  - Provides the base configuration, including the 'poky' distro.
- meta-arm layers (meta-arm-bsp, meta-arm, meta-arm-toolchain)
  - URL: https://git.yoctoproject.org/meta-arm/?h=langdale
  - meta-arm-bsp contains the board support to boot baremetal Linux for the `fvp-baser-aemv8r64` machine. It includes
    - The base device tree
    - Machine-specific boot-wrapper-aarch64 patches
    - Machine-specific U-Boot patches
  - meta-arm contains:
    - A recipe for the FVP_Base_AEMv8R model itself
    - The base recipe for boot-wrapper-aarch64
  - meta-arm-toolchain is not used, but is a dependency of meta-arm.
- meta-virtualization
  - URL: https://git.yoctoproject.org/meta-virtualization/?h=langdale
  - Included for the Virtualization stack only.
- meta-openembedded layers (meta-oe, meta-python, meta-filesystems, meta-networking)
  - URL: http://git.openembedded.org/meta-openembedded?h=langdale
  - These are dependencies of meta-virtualization and meta-zephyr. `meta-oe` and `meta-python` are included for all configurations. `meta-filesystems` and `meta-networking` are included for the Virtualization stack only.
- meta-zephyr layer (meta-zephyr-core)
  - URL: https://git.yoctoproject.org/meta-zephyr/?h=langdale
  - Provides Zephyr SDK bundle, which is used to build Zephyr applications.
  - Provides recipes for `zephyr-helloworld`, `zephyr-synchronization` and `zephyr-philosophers`.

- meta-openamp
  - URL: https://github.com/OpenAMP/meta-openamp
  - Included for the Virtualization stack only.
  - Provides support for building libmetal and libopen_amp libraries.

3.4 Armv8-R AArch64 Extras Layer (meta-armv8r64-extras)

3.4.1 Zephyr Sample Applications

This Software Stack supports four Zephyr sample applications:

- zephyr-helloworld
- zephyr-synchronization
- zephyr-philosophers
- zephyr-rpmsg-demo (Zephyr part of *RPMsg Demo*)

Recipes for `zephyr-helloworld`, `zephyr-synchronization` and `zephyr-philosophers` are provided by `meta-zephyr`. Source code for these three sample applications are provided by Zephyr, with extra configuration provided for `zephyr-philosophers` by file meta-armv8r64-extras/dynamic-layers/zephyrcore/recipes-kernel/zephyr-kernel/zephyr-philosophers.bbappend. These three applications can work in both the baremetal stack and the Virtualization stack.

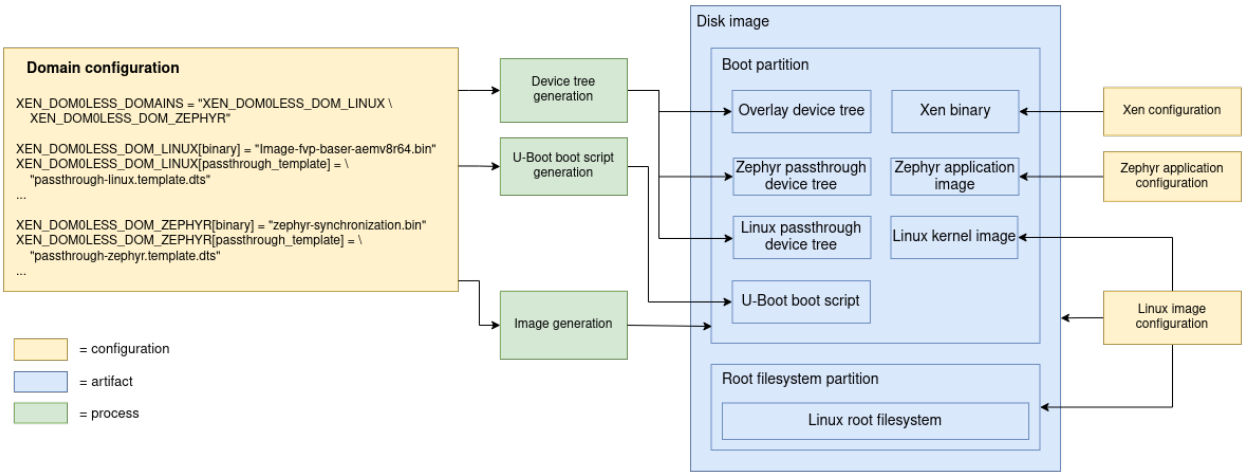
The zephyr-rpmsg-demo application works in the Virtualization stack only. It comes from the recipe meta-armv8r64-extras/dynamic-layers/virtualization-layer/recipes-kernel/zephyr-kernel/zephyr-rpmsg-demo.bb. This application works as a RPMsg host (`rpmsg-host`) in the Zephyr domain, and co-works with the RPMsg remote application (`rpmsg-remote`) running in the Linux domain to implement the Inter-VM communication through the RPMsg protocol. Recipe for the `rpmsg-remote` application is provided by meta-armv8r64-extras/dynamic-layers/virtualization-layer/recipes-demo/rpmsg-demo/rpmsg-demo.bb, which along with `zephyr-rpmsg-demo.bb` is provided by this Software Stack. The source code is also provided by this Software Stack and can be found in directory components/apps/rpmsg-demo. For more details, see *RPMsg Demo* under *Demo Applications*.

Running Zephyr applications is supported by the `runfvp` script from meta-arm. Section *Build and Run* has examples that uses this script.

3.4.2 Virtualization Stack

The Virtualization stack uses Xen as the Type-1 hypervisor to boot one or more domains independently. The meta-virtualization layer is used to provide Xen build support.

The Xen MPU implementation only supports "dom0less" mode (for more details see the *Hypervisor (Xen)* section), so the meta-armv8r64-extras layer provides logic to build the required dom0less artifacts, which are shown in the diagram below. For more information on how U-Boot detects and uses these artifacts at runtime, see the U-Boot *Boot Sequence*.



Domain Configuration

The default domain configuration is provided in a bbclass. For more information, see *Customizing the Xen Domains*.

Xen Configuration

Xen is configured through the `.bb` and `.bbappend` files in directory meta-armv8r64-extras/dynamic-layers/virtualization-layer/recipes-extended/xen, which define the branch and revision to build for the `fvp-baser-aemv8r64` machine and also includes a machine-specific config file.

Linux Image Configuration

In projects based on OE-core, the Linux image recipe is responsible for creating the disk image, including the root filesystem partition. The boot partition, which is part of the same disk image, is populated by extending `IMAGE_CLASSES` (see *Image Generation* below).

No changes are made to the Linux kernel for the Virtualization use case.

Zephyr Application Configuration

Zephyr requires some specific configuration when running as a Xen domain on the `fvp-baser-aemv8r64` machine. It is necessary to match the number of CPUs allocated, the DRAM address and the selected UART with the equivalent parameters in the domain configuration. This is achieved using Xen-specific overlay files (see *Customizing the Zephyr Configuration*).

Device Tree Generation

Xen domains in dom0less mode are configured using additions to the `/chosen` node in the device tree. To avoid modifying the firmware's device tree for the Virtualization stack we instead dynamically generate a device tree overlay file which can be applied at runtime by U-Boot. Additionally, in order for Xen to access peripherals in dom0less mode, we must specify which peripherals to "pass through" to each domain using a domain-specific passthrough dtb file (see https://xenbits.xen.org/docs/unstable/misc/arm/passthrough.txt for more details).

All these device trees are created from templates, substituting placeholders with values defined in the domain configuration. The passthrough device tree template to use for each domain is selected in the domain configuration.

U-Boot Script Generation

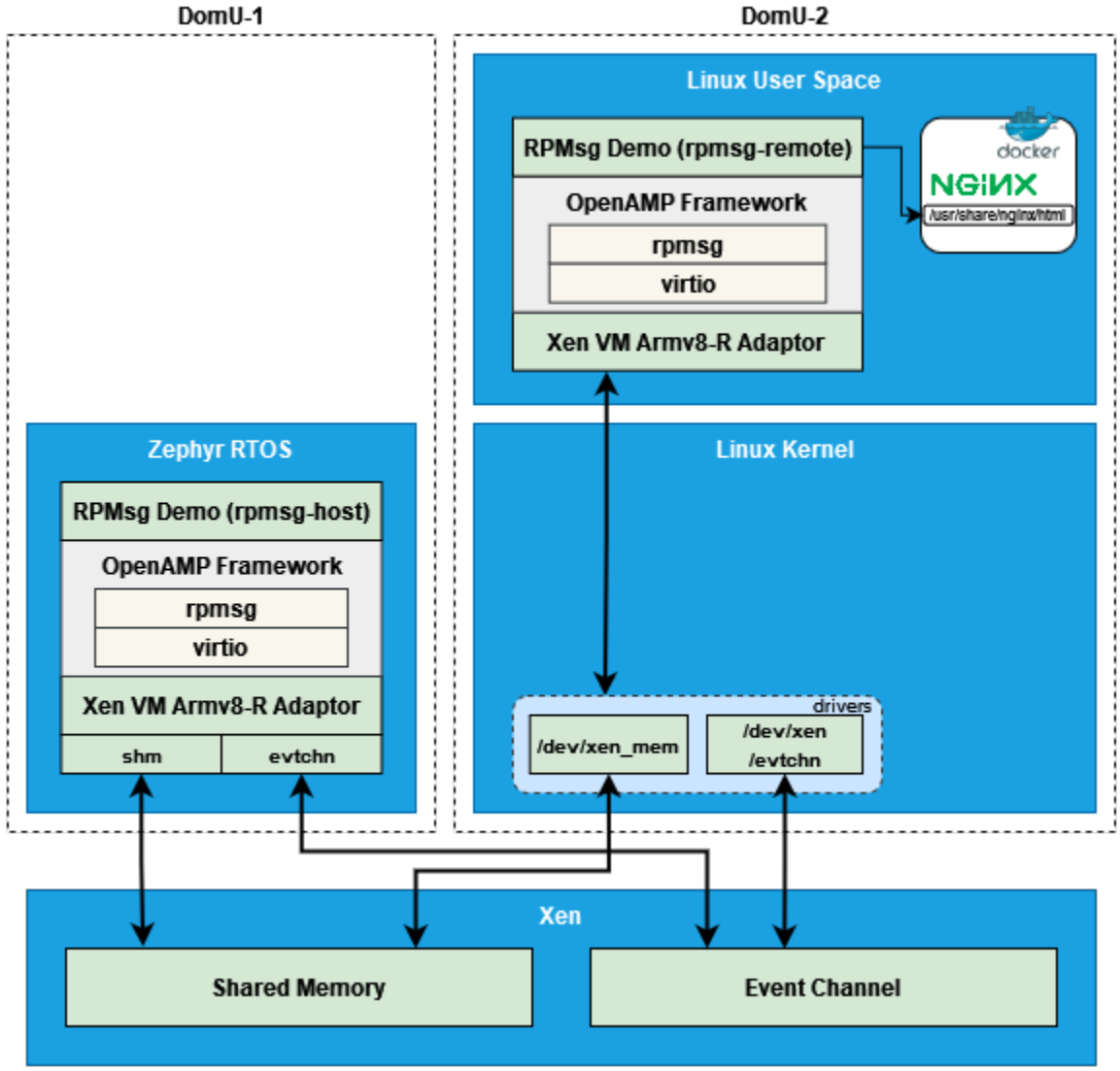
The boot script is partially dynamically generated in order to load the configured domain binaries and passthrough device trees to the configured memory addresses prior to booting Xen.

Image Generation

The boot partition needs to be populated with the artifacts necessary to boot Xen and its domains so they can be loaded by U-Boot. This is achieved using a custom Wic image and a custom bbclass (`xen_image_dom0less`). This class can be added to `IMAGE_CLASSES` so that the desired Linux image recipe (e.g. `core-image-minimal`) includes the necessary configuration to populate the boot partition.

3.5 Applications

The following diagram shows the architecture of the applications implemented in this Software Stack:



3.5.1 Inter-VM Communication

As described in the above architecture diagram, Inter-VM communication implemented in this Software Stack is based on a shared memory mechanism to transfer data between VMs. It is implemented using the OpenAMP framework in the application layer. The Xen hypervisor provides a communication path using static shared memory and static event channel for data transfer. And the guest OSes (Linux and Zephyr) in the middle expose the static shared memory and the static event channel to upper layer applications.

For the implementation of the static shared memory and the static event channel in Xen and 2 guest OSes, please refer to the section *Hypervisor (Xen)*, *Linux Kernel* and *Zephyr* in *Components*.

RPMsg (Remote Processor Messaging) is a messaging protocol enabling communication between two VMs, which can be used by Linux as well as real-time OSes. RPMsg implementation in the OpenAMP library is based on virtio.

This Software Stack introduces the meta-openamp layer to provide support for the OpenAMP library.

3.5.2 Docker Container

The support for docker container in this Software Stack is provided by docker-ce recipe in the meta-virtualization layer.

Running docker also requires `kernel-module-xt-nat`, which is enabled by meta-armv8r64-extras/dynamic-layers/virtualization-layer/recipes-containers/docker/docker-ce_git.bbappend.

3.5.3 Demo Applications

There are two demo applications in this Software Stack to demonstrate these two use scenarios described in the *High Level Architecture* section in *Introduction*.

- RPMsg Demo (components/apps/rpmsg-demo)
- Docker Container Hosted Nginx (meta-armv8r64-extras/dynamic-layers/virtualization-layer/recipes-demo/nginx-docker-demo)

These two applications can work together to complete the following process:

1. The program running in Zephyr periodically collects the data of the system running status
2. Zephyr sends sampled data to Linux via RPMsg
3. After receiving the data, Linux stores the data in a local file
4. The Nginx web server running in a docker container serves this file for external users visiting through the HTTP protocol
5. Repeat the above steps so that users can get continuously updated system running status

RPMsg Demo

The `RPMsg Demo` application consists of two parts:

- `rpmsg-host` runs in the Zephyr domain to sample the data of the system running status and sends the data to `rpmsg-remote`
- `rpmsg-remote` runs in the Linux domain to receive the data and stores it in a local file `/usr/share/nginx/html/zephyr-status.html`

These two parts communicate with each other using the OpenAMP framework in the application layer, and the static shared memory and static event channel provided by Xen in the lower layer.

The recipes for this demo application are provided by meta-armv8r64-extras/dynamic-layers/virtualization-layer/recipes-kernel/zephyr-kernel/zephyr-rpmsg-demo.bb and meta-armv8r64-extras/dynamic-layers/virtualization-layer/recipes-demo/rpmsg-demo/rpmsg-demo.bb. The source code can be found in directory components/apps/rpmsg-demo.

Docker Container Hosted Nginx

In the `Docker Container Hosted Nginx` demo application, the Nginx server serves the file `/usr/share/nginx/html/zephyr-status.html`, and external users can visit the data in a web browser.

This demo application start automatically by default. To start it manually, run script `/usr/share/nginx/utils/run-nginx-docker.sh` in the Linux domain. See the section *Virtualization* in *Reproduce* of *User Guide* for example usage.

The recipe for this demo application is provided by meta-armv8r64-extras/dynamic-layers/virtualization-layer/recipes-demo/nginx-docker-demo/nginx-docker-demo.bb.

Run Demo Applications

In the default configuration, the above two demo applications run automatically after the system starts. See the section *Virtualization* in *Reproduce* of *User Guide* for example usage.

To prevent them from running automatically in the Linux domain, set `XEN_DOM0LESS_DOM_LINUX_DEMO_AUTORUN` to `0` when build, For example:

```
# Build
XEN_DOM0LESS_DOM_LINUX_DEMO_AUTORUN=0 \
    kas build v8r64/meta-armv8r64-extras/kas/virtualization.yml
```

And run the demo applications manually using the following commands in the Linux domain:

```
# Start Nginx web server
/usr/share/nginx/utils/run-nginx-docker.sh

# Start rpmsg-demo
rpmsg-remote
```

Limitations and Improvements

The shared memory and event channel mechanisms that Inter-VM communication relies on are still evolving in Xen, Linux, and Zephyr, which leads to the limitation of the `RPMsg Demo` application.

The `RPMsg Demo` application is mainly to demonstrate the communication between VM guests within the Xen hypervisor. This program does not have a sophisticated fault tolerance and exception recovery mechanism. If an exception occurs, in the extreme case it may be necessary to restart FVP for the next demonstration, especially in the case of running it manually.

In the current implementation, at least the following improvements can be made at the application level:

- In `rpmsg-host`, when `send_message` fails, the current operation is to exit the loop directly. The improvement here can be to add a retry mechanism. If it continues to fail, it can further fall back to trying to

restart `rpmsg_init_vdev` to setup a new RPMsg channel. The detailed code is in components/apps/rpmsg-demo/demos/zephyr/rpmsg-host/src/main.c.

- Implement `rpmsg-host` based on Zephyr IPC subsystem RPMsg Service, which can support multiple endpoints so that it can support multiple RPMsg channels. The reference code can be found here.

3.6 Validation

3.6.1 Run-Time Integration Tests

The run-time integration tests are a mechanism for validating the Software Stack's core functionalities.

The tests are run using the OEQA test framework. Please refer to OEQA on Arm FVPs for more information on the this framework. Some tests are also built as a Yocto Ptest (Package Test) and implemented using the Bash Automated Testing System (BATS).

The integration tests run on an image and depend on its target sub-stack (**Baremetal Zephyr**, **Baremetal Linux** or **Virtualization**). The tests may also change their behaviour depending on the the build parameters for target image.

To run the tests via kas, please refer to the section *Validate* in *User Guide*.

3.6.2 Test Suite

The 3 supported sub-stacks of the Software Stack that are tested by the framework are detailed below. The testing scripts can be found in meta-armv8r64-extras/lib/oeqa/runtime/cases.

Detailed below is a brief description of what each test does:

- zephyr_v8r64

  Zephyr test case which is capable of validating the output of `zephyr-helloworld`, `zephyr-philosophers` and `zephyr-synchronization`.

- linuxboot

  This case is implemented in `meta-arm/lib/oeqa/runtime/cases/linuxboot.py`. It waits for a Linux login prompt on the default console.

- sysinfo

  Test case to check system information (e.g. kernel version, hostname) is the same between Yocto build system and Yocto image.

- basictests

  Run `basic-tests` implemented using BATS. See *Basic Tests* for more details.

- xen

  Test case which is capable of validating the boot of Xen.

- ptest_docker

  Test case to check docker engine installation, daemon and runtime.

- rpmsg

  Test case to validate the Inter-VM communication using RPMsg between the Zephyr domain and the Linux domain on Xen.

The following table is a map of the test sets contained in the 3 sub-stacks:

Test Case	Baremetal Zephyr	Baremetal Linux	Virtualization
zephyr_v8r64	x		x
linuxboot		x	x
sysinfo		x	x
basictests		x	x
linuxlogin			x
xen			x
ptest_docker			x
rpmsg			x

For the Virtualization stack, more specifically, whether the test case is applicable or not depends on the parameters to the build the target image, and the corresponding relationship is as follows:

Test Case	Two DomUs (default) XEN_DOM0LESS_DOMAINS: • ZEPHYR + LINUX ZEPHYR_APPLICATION: • zephyr-rpmsg-demo	Two DomUs XEN_DOM0LESS_DOMAINS: • ZEPHYR + LINUX ZEPHYR_APPLICATION: • zephyr-helloworld • zephyr-synchronization • zephyr-philosophers	Single DomU (Zephyr) XEN_DOM0LESS_DOMAINS: • ZEPHYR ZEPHYR_APPLICATION: • zephyr-helloworld • zephyr-synchronization • zephyr-philosophers	Single DomU (Linux) XEN_DOM0LESS_DOMAINS: • LINUX
zephyr_v8r64	x	x	x	
linuxboot	x	x		x
sysinfo	x	x		x
basictests	x	x		x
linuxlogin	x	x		x
xen	x	x	x	x
ptest_docker	x	x		x
rpmsg	x			

For the build parameters `XEN_DOM0LESS_DOMAINS` and `XEN_DOM0LESS_ZEPHYR_APPLICATION`, please refer to the section *Customizing Build Environment Parameters* in *User Guide*.

3.6.3 Basic Tests

The Basic Tests perform the basic checks on the devices and functionalities supported in the system. These tests are available for both the **Baremetal Linux** and **Virtualization** images and consist of a series of BATS tests that can be found in meta-armv8r64-extras/recipes-refstack-tests/basic-tests/files/testcases.

Detailed below is information on which test is applicable to which stack and a brief description of what each test does:

- Baremetal Linux:
  - rtc

The BATS script implementing the test is `test-rtc.bats`. Checks that the rtc (real-time clock) device and its correct driver are available and accessible via the filesystem and verifies that the `hwclock` command runs successfully.

- watchdog

  The BATS script implementing the test is `test-watchdog.bats`. Checks that the watchdog device and its correct driver are available and accessible via the filesystem.

- networking

  The BATS script implementing the test is `test-net.bats`. Checks that the network device and its correct driver are available and accessible via the filesystem and that outbound connections work (invoking `wget`).

- smp

  The BATS script implementing the test is `test-smp.bats`. Checks for CPU availability and that basic functionality works, like enabling and stopping CPUs and preventing all of them from being disabled at the same time.

- virtiorng

  The BATS script implementing the test is `test-virtiorng.bats`. Check that the virtio-rng device is available through the filesystem and that it is able to generate random numbers when required.

- Virtualization:
  - Linux Domain
    - Same tests as Baremetal Linux
    - shmem

      The BATS script implementing the test is `test-shmem.bats`. Check that the `xen_mem` device is available.

    - evtchn

      The BATS script implementing the test is `test-evtchn.bats`. Check that the `xen/evtchn` device is available.

The Basic Tests are built and installed in the image according to the following BitBake recipe: meta-armv8r64-extras/recipes-refstack-tests/basic-tests/basic-tests.bb.

3.6.4 Integration Tests Implementation

This section gives a high-level description of how the integration testing logic is implemented.

To enable the integration tests, the testimage.bbclass is used. This class supports running automated tests against images. The class handles loading the tests and starting the image.

The Writing New Tests section of the Yocto Manual explains how to write new tests when using the `testimage.bbclass`. These are placed under meta-armv8r64-extras/lib/oeqa/runtime/cases and will be selected by the different machines/configurations by modifying the `TEST_SUITES` variable. For example, the file meta-armv8r64-extras/classes/refstack-virtualization-tests.bbclass adds different combinations of test cases to the `TEST_SUITES` variable according to the build parameters of the virtualizaiton stack.

For the Basic Tests, the `basic-tests.bb` that inherits the Ptest framework/class and installs the BATS test files located under meta-armv8r64-extras/recipes-refstack-tests/basic-tests. Each image recipe (**Baremetal Linux** or **Virtualization**) includes `basic-tests.bb`.

CHAPTER

FOUR

CODELINE MANAGEMENT

4.1 Overview

The Armv8-R AArch64 Software Stack releases use the following model:

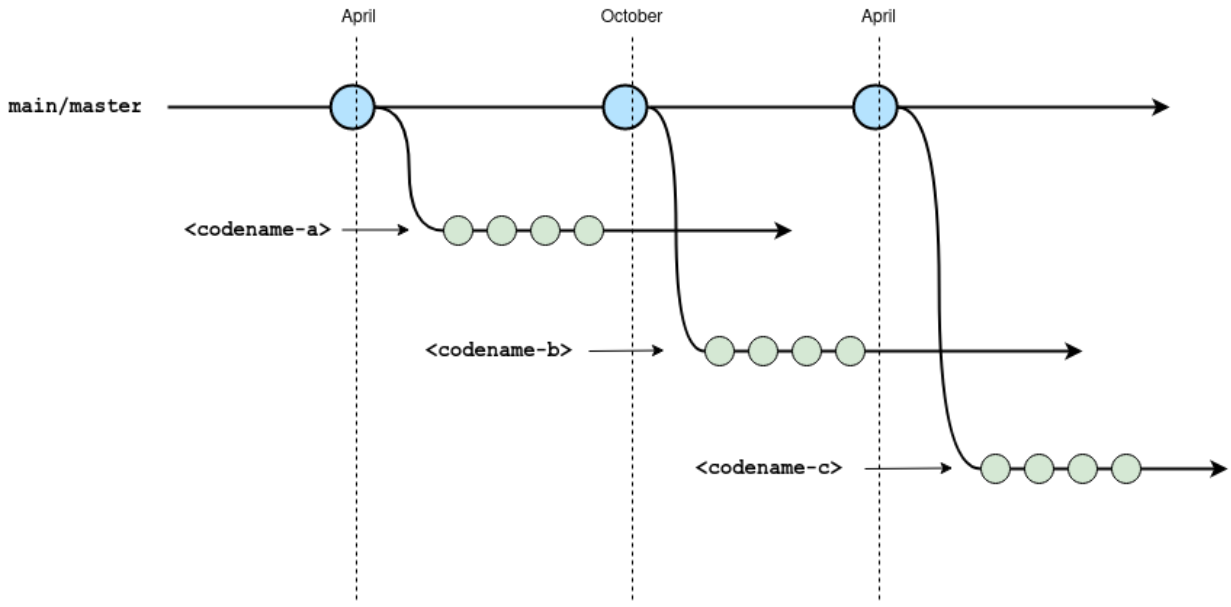
- The Software Stack releases grow incrementally (meaning that we are not dropping previously released features, with exceptions)

  Note: A specific exception is that support for PREEMPT_RT Linux builds was removed in *Release 4.0 - Xen*.

- The Armv8-R AEM FVP model releases are incremental (meaning that FVPs are not dropping previously released features)

In addition to the above model, the Armv8-R AArch64 Software Stack is developed and released based on Yocto's release branch process. This strategy allows us make releases based on upstream stable branches, reducing the risk of having build and runtime issues.

4.2 Yocto Release Process



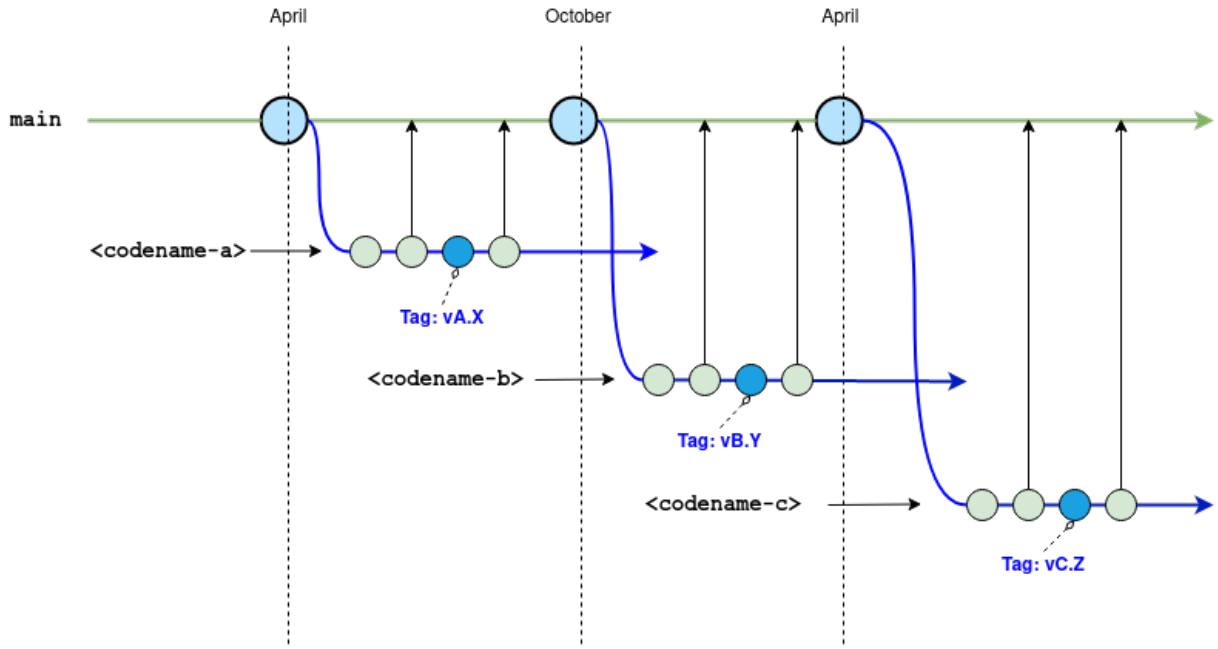
The diagram above gives an overview of the Yocto branch and release process:

- Development happens primarily in the `master` (or `main`) branch.
- The project has a major release roughly every 6 months where a stable branch is created.
- Each major release has a *codename* which is also used to name the stable branch is created.
- Once a stable branch is created and released, it only receives bug fixes with minor (point) releases on an unscheduled basis.
- The goal is for users and 3rd parties layers to use these *codename* branches as a means to be compatible with each other.

For a complete description of the Yocto release process, support schedule and other details, see the Yocto Release Process documentation.

4.3 Armv8-R AArch64 Software Stack Branch and Release Process

The Armv8-R AArch64 Software Stack's branch and release process can be described as the following diagram:



The stack's branch and release process will follow the Yocto release process. Below is a detailed description of the branch strategy for this stack's development and release process.

- Main branch
  - Represented by the green line on the diagram above.
  - The repository's `main` branch is meant to be compatible with `master` or `main` branches from Poky and 3rd party layers.
  - This stack is not actively developed on this `main` branch to avoid the instability inherited from Yocto development on the `master` branch.
  - To reduce the effort to move this stack to a new version of Yocto, this `main` branch is periodically updated with the patches from the development and release branch on a regular basis.

- Development and release branches
  - Represented by the blue line on the diagram above.
  - This stack uses development branches based on or compatible with Yocto stable branches.
  - A development branch in this stack is setup for each new Yocto release using the name convention *codename* where *codename* comes from target Yocto release.
  - The development branches are where fixes, improvements and new features are developed.
  - On a regular basis, code from the development branch is ported to the `main` branch to reduce the effort required to move this stack to a new version of Yocto.
  - The development branches are also used for release by creating tags on the specific commits.

CHAPTER

FIVE

LICENSE

The software is provided under the MIT license (below).

```
Copyright (c) <year> <copyright holders>

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to
deal in the Software without restriction, including without limitation the
rights to use, copy, modify, merge, publish, distribute, sublicense, and/or
sell copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice (including the next
paragraph) shall be included in all copies or substantial portions of the
Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS
IN THE SOFTWARE.
```

5.1 SPDX Identifiers

Individual files contain the following tag instead of the full license text.

```
SPDX-License-Identifier: MIT
```

This enables machine processing of license information based on the SPDX License Identifiers that are here available: http://spdx.org/licenses/

CHAPTER

SIX

CHANGELOG & RELEASE NOTES

6.1 Release 5.0 - InterVM Communication with Linux Containers

6.1.1 New Features

- Provided static shared memory in Xen hypervisor allowing users to statically set up shared memory on a dom0less system, enabling domains to do shm-based communication
- Provided static event channel established in Xen hypervisor for event notification between two domains
- Implemented Inter-VM communication using RPMsg protocol provided by the OpenAMP framework together with the static shared memory and static event channel provided by Xen hypervisor
- Added docker support in the Linux domain, showing standard application deployments via containers
- Added a docker container hosted Nginx web server to serve data for external users visiting through the HTTP protocol
- Added Yocto testimage support
- Added test suite that can validate the functionalities in Baremetal Zephyr
- Enhanced the test suite to validate Baremetal Linux and Virtualization stack

6.1.2 Changed

- Upgraded to FVP version 11.20.15
- Upgraded Yocto to langdale
- Upgraded to U-Boot 2022.07
- Upgraded to Xen 4.17
- Upgraded to Zephyr 3.2.0
- Upgraded to Linux kernel 5.19
- Enabled `CONFIG_XEN` options for Linux domain to add Xen kernel support
- Changed to use the meta-zephyr Yocto layer to build Zephyr applications

```
URL: https://git.yoctoproject.org/meta-zephyr
layers: meta-zephyr-core
branch: langdale
revision: 375aea434ef214a5022e0a3e54fcae3e5e965c9b
```

- Introduced the meta-openamp Yocto layer to build libmetal and OpenAMP libraries

```
URL: https://github.com/OpenAMP/meta-openamp
layers: meta-openamp
|inclusivity-exception| branch: master
revision: cd5cc9274321ea9d808a7800cfcba26320cf53a5
```

- Other third-party Yocto layers used to build the Software Stack:

```
URL: https://git.yoctoproject.org/git/poky
layers: meta, meta-poky
branch: langdale
revision: a3e3b740e140d036122f7b11e2ac452bda548444

URL: https://git.openembedded.org/meta-openembedded
layers: meta-filesystems, meta-networking, meta-oe, meta-python
branch: langdale
revision: c354f92778c1d4bcd3680af7e0fb0d1414de2344

URL: https://git.yoctoproject.org/git/meta-virtualization
layers: meta-virtualization
branch: langdale
revision: 8857b36ebfec3d548755755b009adc491ef320ab

URL: https://git.yoctoproject.org/git/meta-arm
layers: meta-arm, meta-arm-bsp, meta-arm-toolchain
branch: langdale
revision: 025124814e8676e46d42ec5b07220283f1bdbcd0
```

6.1.3 Known Issues and Limitations

- Xen `shared-memory` does not support `xen,offset` yet, which is introduced in the Linux documentation at Xen hypervisor reserved-memory binding
- The below feature for Xen static shared memory device node "xen,shared-mem" is not implemented yet:

  Host physical address is optional, when missing Xen decides the location

- The limitations of *RPMsg Demo* mentioned in *Limitations and Improvements*
- The `bp.refcounter.use_real_time` model parameter is not supported for the hypervisor and baremetal Zephyr use cases
- Issues and limitations mentioned in *Known Issues and Limitations of Release 2*

6.2 Release 4.0 - Xen

6.2.1 New Features

- Added ability to boot one Linux and one Zephyr domain on Xen in dom0less mode
- Added ability to boot three Zephyr sample applications
- Added basic Linux test suite using BATS

6.2.2 Changed

- A new Yocto layer has been created called `meta-armv8r64-extras`, which extends `meta-arm-bsp` with additional use cases
- Moved documentation to the new repository and expanded to cover new use cases. It is now published on *ReadTheDocs*
- Upgraded to FVP version 11.18.16
- Upgraded to U-Boot 2022.01 and enabled support for applying device tree overlays
- Upgraded to Linux kernel 5.15
- Added support to boot-wrapper-aarch64 for S-EL2 SMP payloads such as Xen
- Amended the device tree to support the virtio random number generator
- Added *ssh-pregen-hostkeys* to the default image to improve boot times
- Amended the default model parameters to support enabling `cache_state_modelled`
- Removed support for PREEMPT_RT Linux kernel builds
- Third-party Yocto layers used to build the Software Stack:

```
URL: https://git.yoctoproject.org/git/poky
layers: meta, meta-poky
branch: kirkstone
revision: 0c3b92901cedab926f313a2b783ff2e8833c95cf

URL: https://git.openembedded.org/meta-openembedded
layers: meta-filesystems, meta-networking, meta-oe, meta-python
branch: kirkstone
revision: fcc7d7eae82be4c180f2e8fa3db90a8ab3be07b7

URL: https://git.yoctoproject.org/git/meta-virtualization
layers: meta-virtualization
branch: kirkstone
revision: 0d35c194351a9672d18bff52a8f2fbabcd5b0f3d

URL: https://git.yoctoproject.org/git/meta-arm
layers: meta-arm, meta-arm-bsp, meta-arm-toolchain
branch: kirkstone
revision: 32ca791aaa6634e90a7c6ea3994189ef10a7ac90
```

6.2.3 Known Issues and Limitations

- The `bp.refcounter.use_real_time` model parameter is not supported for the hypervisor and baremetal Zephyr use cases
- "Dom0less" is a new set of features for Xen, some of which are still being developed in the Xen community. In this case, a "dom0less" domain running Linux cannot enable the `CONFIG_XEN` option, which will cause some limitations. For example, Linux can't detect that it is running inside a Xen domain, so some Xen domain platform-specific Power Management control paths will not be invoked. One of the specific cases is that using the command `echo 0 > /sys/devices/system/cpu/cpu0/online` to power off the Linux domain's vcpu0 is invalid.
- Issues and limitations mentioned in *Known Issues and Limitations of Release 2*

6.3 Release 3 - UEFI

6.3.1 Release Note

https://community.arm.com/oss-platforms/w/docs/638/release-3—uefi

6.3.2 New Features

- Added U-Boot v2021.07 for UEFI support
- Updated boot-wrapper-aarch64 revision and added support for booting U-Boot
- Included boot-wrapper-aarch64 PSCI services in `/memreserve/` region

6.3.3 Changed

- Configured the FVP to use the default RAM size of 4 Gb
- Added virtio_net User Networking mode by default and removed instructions about tap networking setup
- Updated Linux kernel version from 5.10 to 5.14 for both standard and Real-Time (PREEMPT_RT) builds
- Fixed the counter frequency initialization in boot-wrapper-aarch64
- Fixed `PL011` and `SP805` register sizes in the device tree

6.3.4 Known Issues and Limitations

- Device DMA memory cache-coherence issue: the FVP `cache_state_modelled` parameter will affect the cache coherence behavior of peripherals' DMA. When users set `cache_state_modelled=1`, they also have to set `cci400.force_on_from_start=1` to force the FVP to enable snooping on upstream ports
- Issues and limitations mentioned in *Known Issues and Limitations of Release 2*

6.4 Release 2 - SMP

6.4.1 Release Note

https://community.arm.com/oss-platforms/w/docs/634/release-2—smp

6.4.2 New Features

- Enabled SMP support via boot-wrapper-aarch64 providing the PSCI `CPU_ON` and `CPU_OFF` functions
- Introduced Armv8-R64 compiler flags
- Added Linux PREEMPT_RT support via linux-yocto-rt-5.10
- Added support for file sharing with the host machine using Virtio P9
- Added support for runfvp
- Added performance event support (PMU) in the Linux device tree

6.4.3 Changed

None.

6.4.4 Known Issues and Limitations

- Only PSCI `CPU_ON` and `CPU_OFF` functions are supported
- Linux kernel does not support booting from secure EL2 on Armv8-R AArch64
- Linux KVM does not support Armv8-R AArch64

6.5 Release 1 - Single Core

6.5.1 Release Note

https://community.arm.com/oss-platforms/w/docs/633/release-1-single-core

6.5.2 New Features

Introduced fvp-baser-aemv8r64 as a Yocto machine support the following BSP components on the Yocto hardknott release branch, where a standard Linux kernel can be built and run:

- boot-wrapper-aarch64
- Linux kernel: linux-yocto-5.10

6.5.3 Changed

Initial version.

6.5.4 Known Issues and Limitations

- Only support one CPU since SMP is not functional in boot-wrapper-aarch64 yet