# Armv8-R AArch64 Software Stack

Arm Ltd.

Jan 11, 2023

# CONTENTS

CHAPTER

ONE

# INTRODUCTION

## 1.1 Overview

The Armv8-R AArch64 software stack is a collection of example software compositions for the Armv8-R AEM FVP model[1], an implementation of the Armv8-R AArch64 architecture[2]. The Open Source example software coupled with the Armv8-R AEM FVP enable the exploration of the general Armv8-R AArch64 architecture as well as its specific implementation - Cortex-R82[3], where FVP stands for Fixed Virtual Platforms, and AEM stands for Architecture Envelope Model.

This software stack supports two types of operating system: Linux as the Rich OS and Zephyr as the RTOS (Real-Time Operating System), and can run either on baremetal (without hardware virtualization) or virtualization with Xen as Type-1 hypervisor. It is provided via a set of Yocto layers which contains all the instructions necessary to fetch and build the source as well as to download the model and launch the example.

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:

- Arm architectures[7]

[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
[7] https://developer.arm.com/architectures

- 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 configurations supported in the software stack for either baremetal solution or virtualization solution separately:

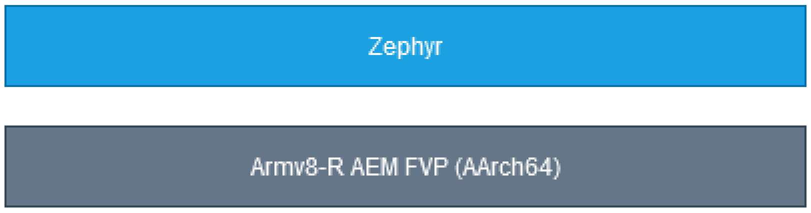
- Baremetal solution (without hardware virtualization support)
  - Boot a Linux based Rich OS (via U-boot with UEFI) to command prompt
  - Boot Zephyr RTOS directly and run a sample application
- Virtualization solution
  - Boot the Xen hypervisor (via U-boot without UEFI) and start two isolated domains to run Linux Rich OS and Zephyr RTOS in parallel on the fixed number of cores assigned statically by the Xen hypervisor

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

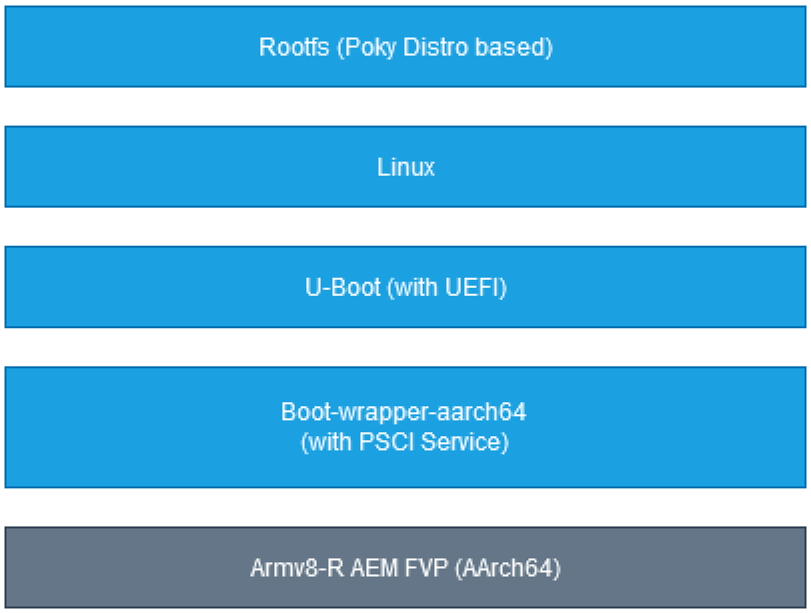
The high-level architecture corresponding to the three use cases (as described in the *Use Cases Overview* section above) supported by this software stack is as follows:

The diagram below gives an overview of the components and layers in the baremetal Zephyr implementation in this stack.

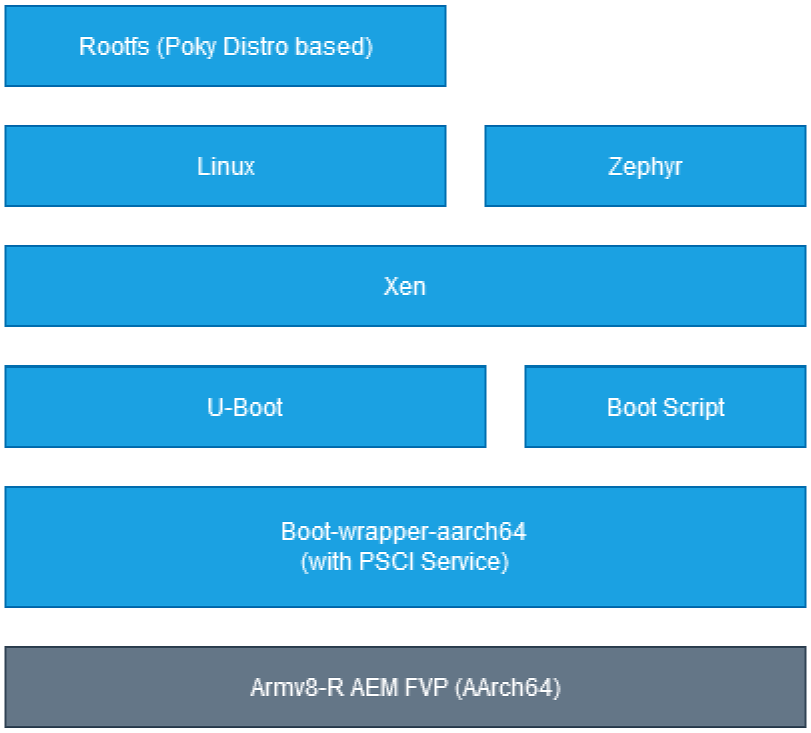


The diagram below gives an overview of the components and layers in the baremetal Linux implementation in this stack.

[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/


Rootfs (Poky Distro based)
Linux
U-Boot (with UEFI)
Boot-wrapper-aarch64 (with PSCI Service)
Armv8-R AEM FVP (AArch64)

The diagram below gives an overview of the components and layers in the implementation of virtualization with Xen hypervisor in this stack.


Rootfs (Poky Distro based)
Linux
Zephyr
Xen
U-Boot
Boot Script
Boot-wrapper-aarch64 (with PSCI Service)
Armv8-R AEM FVP (AArch64)

## 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
  - Test suite that can validate the basic functionalities in Linux
  - 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

## 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.

- *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.

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[18] via GitLab[13], following the project's Issue template.

[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=kirkstone
[16] https://git.yoctoproject.org/meta-arm/
[17] https://www.sphinx-doc.org/
[18] https://gitlab.arm.com/automotive-and-industrial/v8r64/-/issues

## 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)

- Diego Sueiro <diego.sueiro@arm.com>
- 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/4.0.1/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`, install it like so:

```
pip3 install --user --upgrade kas==3.0.2
```

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 50 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 v4.0
```

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

### 2.1.3 Build and Run

The software stack supports building and running three configurations: 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 configurations are as below:

#### Baremetal Zephyr

Build and run with the baremetal Zephyr configuration:

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

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

# 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 using the Kas `--target` flag, 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` of which `zephyr-synchronization` is set as the default target. See the *Zephyr* section for more information.

#### Baremetal Linux

Build and run with the baremetal Linux configuration:

```
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 virtualization configuration:

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

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

# Output images will be 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 --console"

# Check guest OS in another terminals after Xen started
# Zephyr
telnet localhost 5001
# Linux (login with root with empty password)
telnet localhost 5002
```

**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.

### 2.1.4 Validate

#### Zephyr

For the Zephyr RTOS (both on baremetal and as a Xen domain), these three sample applications supported in this software stack are themselves test programs.

The steps to build and run the sample applications are described in the *Build and Run* section above.

After Zephyr starts, the sample application runs automatically, and you will see the following output (take `zephyr-synchronization` as an example):

```
*** Booting Zephyr OS build zephyr-v3.1.0  ***
Secondary CPU core 1 (MPID:0x1) is up
thread_a: Hello World from cpu 0 on fvp_baser_aemv8r!
thread_b: Hello World from cpu 1 on fvp_baser_aemv8r!
thread_a: Hello World from cpu 0 on fvp_baser_aemv8r!
thread_b: Hello World from cpu 1 on fvp_baser_aemv8r!
```

**Note:** For Zephyr running as a Xen domain, use command `telnet localhost 5001` in another terminal to check the output.

### Linux

For the Linux OS (both on baremetal and as a Xen domain), the software stack contains a test suite which is defined in kas configuration file `meta-armv8r64-extras/kas/tests.yml`. It can be included into the target build to validate Linux's functionalities as below examples.

To build image with the test suite and run the test suite, use the following commands:

```
# For Linux on baremetal
# Build
kas build v8r64/meta-armv8r64-extras/kas/baremetal-linux.yml:v8r64/meta-armv8r64-extras/
↪kas/tests.yml
# Run
kas shell -k v8r64/meta-armv8r64-extras/kas/baremetal-linux.yml:v8r64/meta-armv8r64-
↪extras/kas/tests.yml \
    -c "../layers/meta-arm/scripts/runfvp --verbose --console"
# Login with root account
-or-
# Remote login using ssh in another terminal
ssh -p 8022 root@localhost
# In FVP target
root@fvp-baser-aemv8r64:~# ptest-runner basic-tests


# For Linux running as a Xen domain
# Build
kas build v8r64/meta-armv8r64-extras/kas/virtualization.yml:v8r64/meta-armv8r64-extras/
↪kas/tests.yml
# Run
kas shell -k v8r64/meta-armv8r64-extras/kas/virtualization.yml:v8r64/meta-armv8r64-
↪extras/kas/tests.yml \
    -c "../layers/meta-arm/scripts/runfvp --verbose --console"
# Login with root account in another terminal
telnet localhost 5002
# login: root
-or-
# Remote login using ssh in another terminal
ssh -p 8022 root@localhost
# In FVP target
root@fvp-baser-aemv8r64:~# ptest-runner basic-tests
```

**Note:** Similar to ports 5000 ~ 5003, if port 8022 is already occupied, it will also be automatically incremented to the next available port. Below is an example where you need to use port 8023 to execute the ssh command.

```
Warning: FVP_BaseR_AEMv8R: bp.virtio_net.hostbridge: Unable to bind to port '8022'
Info: FVP_BaseR_AEMv8R: bp.virtio_net.hostbridge: ...binding to port 8023 instead
```

An example of the running result of the test suite is as follows:

```
root@fvp-baser-aemv8r64:~# ptest-runner basic-tests
START: ptest-runner
2022-06-14T04:11
BEGIN: /usr/lib/basic-tests/ptest
1..6
ok 1 physical network is present
ok 2 physical network device got ip address
ok 3 all CPU cores are up
ok 4 SMP is enabled
ok 5 SMP CPU hot plug
ok 6 removing all cores and will fail when removing the last one
DURATION: 7
END: /usr/lib/basic-tests/ptest
2022-06-14T04:11
STOP: ptest-runner
TOTAL: 1 FAIL: 0
```

### 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].

### 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.

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

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 at the path `v8r64/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.

#### 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 `v8r64/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-synchronization`) 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 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] https://docs.zephyrproject.org/3.1.0/develop/application/index.html#important-build-system-variables

### 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 in Xen's integration/mpu branch initialize the PoC (Proof of Concept) of Xen on the Armv8-R AArch64 architecture. The integration/mpu branch with these patches can be built and run as a standalone project outside of this software stack, to support Xen on the Armv8-R AArch64 architecture. Here is the guidance on how to run it outside of this stack.

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

CHAPTER

# THREE

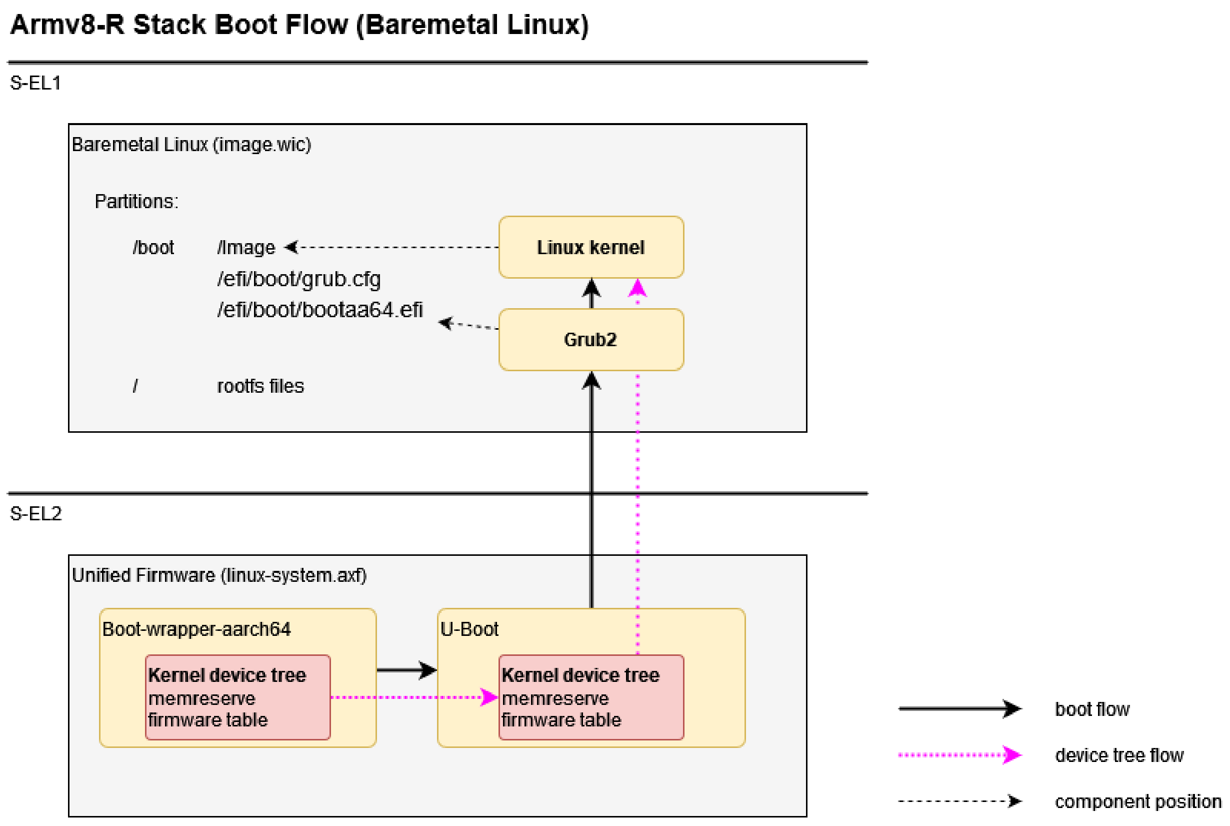
# DEVELOPER MANUAL

## 3.1 Boot Process

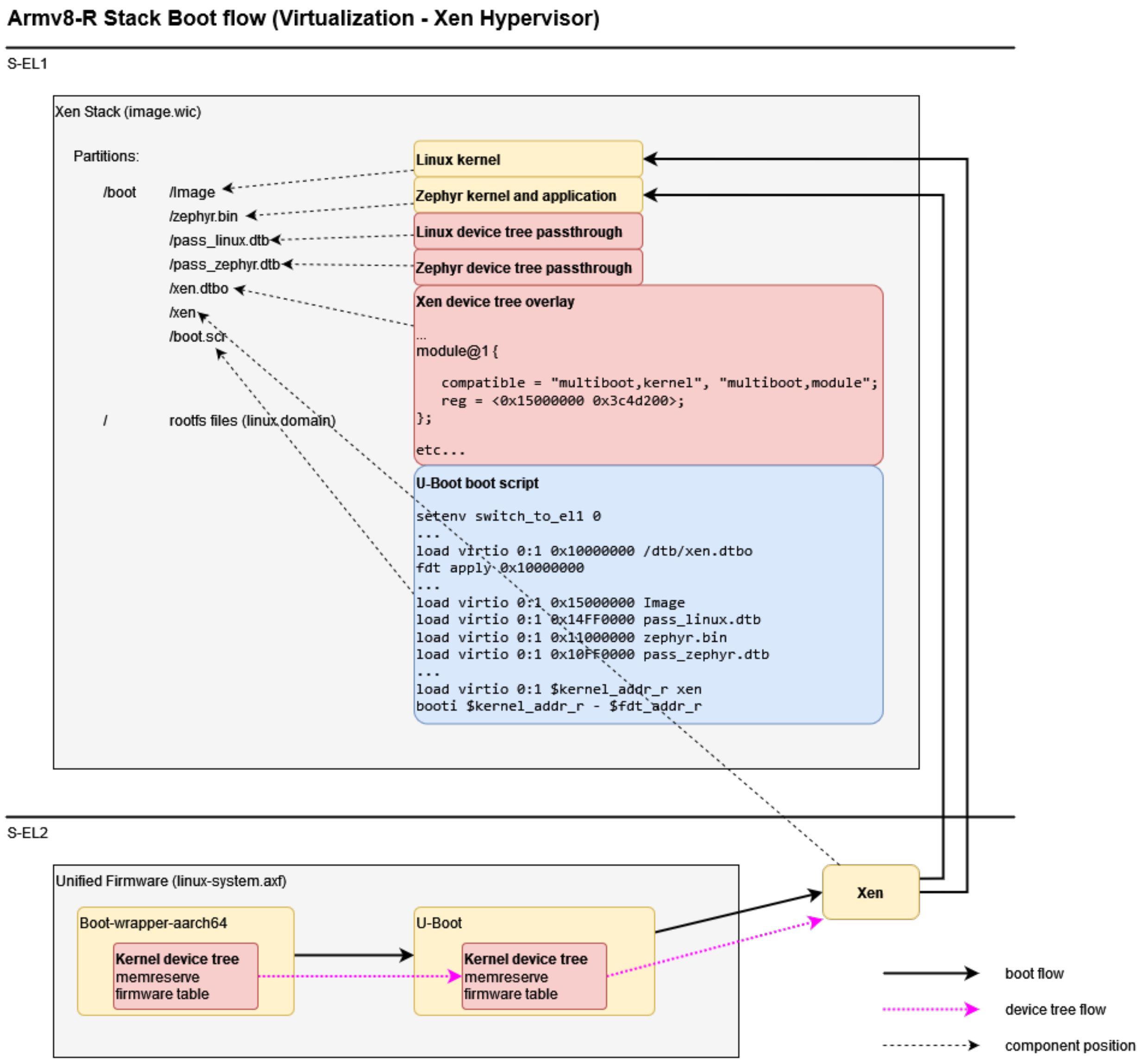
As described in the *High Level Architecture* section of the *Introduction*, the system can boot in 3 different ways. The corresponding boot flow is as follows:

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 in this stack is as the following diagram:


Armv8-R Stack Boot flow (Virtualization - Xen Hypervisor)
S-EL1
Xen Stack (image.wic)
Partitions:
/boot
/Image
/zephyr.bin
/pass_linux.dtb
/pass_zephyr.dtb
/xen.dtbo
/xen
/boot.scr
/
rootfs files (linux domain)
Linux kernel
Zephyr kernel and application
Linux device tree passthrough
Zephyr device tree passthrough
Xen device tree overlay
U-Boot boot script
S-EL2
Unified Firmware (linux-system.axf)
Boot-wrapper-aarch64
Kernel device tree memreserve firmware table
U-Boot
Kernel device tree memreserve firmware table
Xen
boot flow
device tree flow
component position

```
...
module@1 {
    compatible = "multiboot,kernel", "multiboot,module";
    reg = <0x15000000 0x3c4d200>;
};
etc...
```

```
setenv switch_to_el1 0
...
load virtio 0:1 0x10000000 /dtb/xen.dtbo
fdt apply 0x10000000
...
load virtio 0:1 0x15000000 Image
load virtio 0:1 0x14FF0000 pass_linux.dtb
load virtio 0:1 0x11000000 zephyr.bin
load virtio 0:1 0x10FF0000 pass_zephyr.dtb
...
load virtio 0:1 $kernel_addr_r xen
booti $kernel_addr_r - $fdt_addr_r
```

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 2022.01), 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.

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 (because Xen's integration/mpu branch does not support EFI). 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 at commit 3f5d614663 and the patches in integration/mpu branch 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, heap, number of CPU cores and supported devices, etc are all fully static allocated at compile time.

### 3.2.4 Linux Kernel

The Linux kernel in this stack is version 5.15 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

### 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 3.1.0) 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.

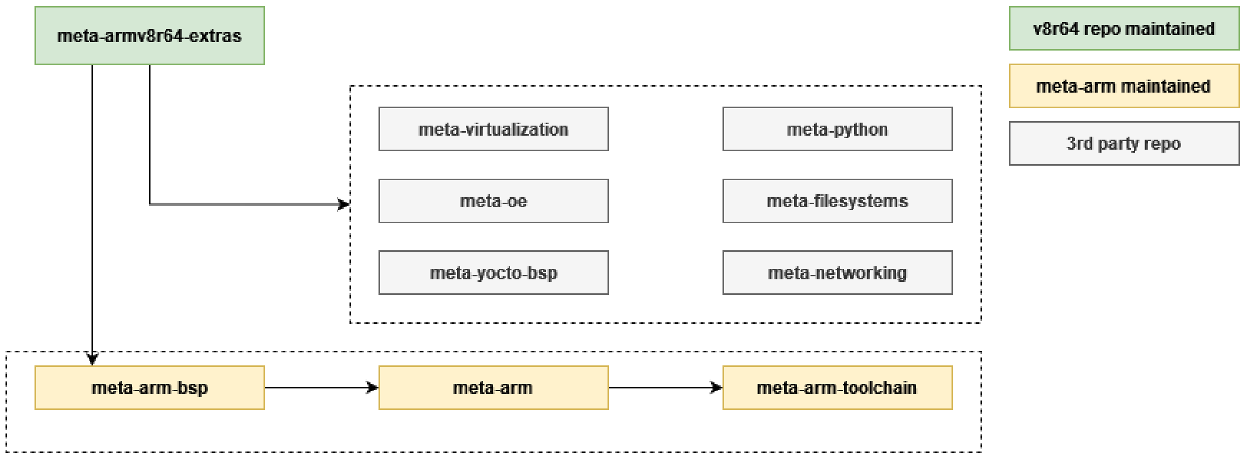
Zephyr provides many demos and sample programs. This software stack supports the following 3 classic samples:

- Hello World
- Synchronization Sample
- Dining Philosophers

## 3.3 Yocto Layers

The Armv8-R AArch64 software stack is provided through the v8r64 repository. 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=kirkstone
  - 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=kirkstone
  - 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=kirkstone
  - 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=kirkstone
  - These are dependencies of meta-virtualization, so are included for the virtualization stack only.

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

### 3.4.1 Zephyr Sample Applications

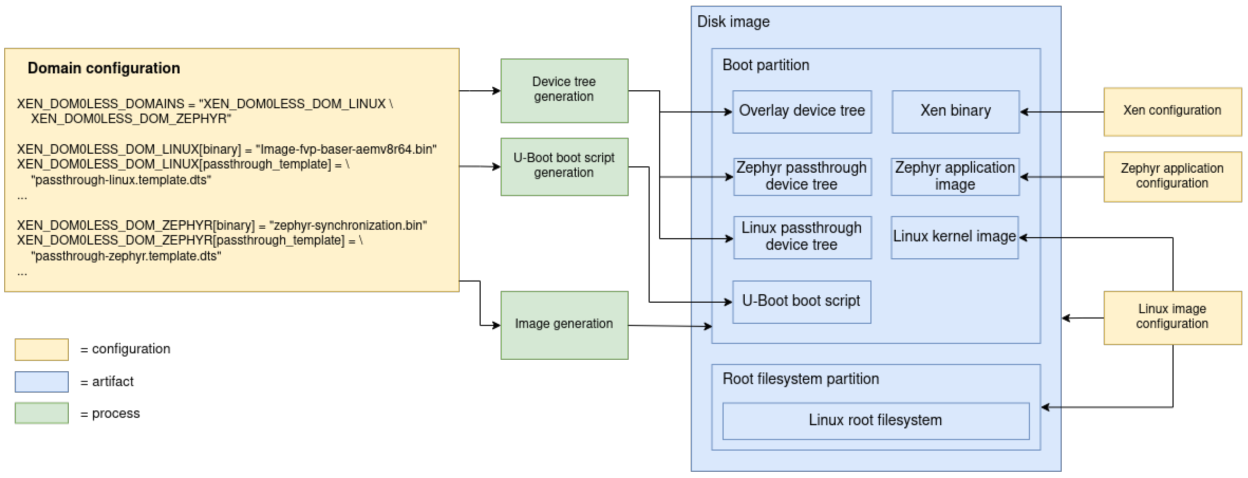
Recipes are provided to build three Zephyr sample applications (`zephyr-helloworld`, `zephyr-synchronization` and `zephyr-philosophers`). These recipes use a common base include file (`zephyr-base.inc`), meaning that application recipes only need to define the path to their source code. There is also support for running Zephyr applications using the `runfvp` script from meta-arm.

The applications are built using the Zephyr SDK, which is pre-compiled and downloaded automatically, so the implementation is standalone and does not depend on meta-zephyr (https://git.yoctoproject.org/meta-zephyr). This means that the generated binary files are identical to those built by following the guidelines in the Zephyr documentation.

### 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**

```
XEN_DOM0LESS_DOMAINS = "XEN_DOM0LESS_DOM_LINUX \
  XEN_DOM0LESS_DOM_ZEPHYR"

XEN_DOM0LESS_DOM_LINUX[binary] = "Image-fvp-baser-aemv8r64.bin"
XEN_DOM0LESS_DOM_LINUX[passthrough_template] = \
  "passthrough-linux.template.dts"
...

XEN_DOM0LESS_DOM_ZEPHYR[binary] = "zephyr-synchronization.bin"
XEN_DOM0LESS_DOM_ZEPHYR[passthrough_template] = \
  "passthrough-zephyr.template.dts"
...
```

#### 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 bbappend file at `v8r64/meta-armv8r64-extras/dynamic-layers/virtualization-layer/recipes-extended/xen`, which defines 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.

### 3.4.3 Test Suite

The test cases in the test suite are implemented based on bats (Bash Automated Testing System) and include 2 categories:

- Network
- SMP

The test suite is added to Yocto Ptest (Package Test) framework and can be run with `ptest-runner` in the target machine.

The test suite can be added into the image by adding the following line in `conf/local.conf`:

```
DISTRO_FEATURES:append = " refstack-tests"
```

CHAPTER

FOUR

# CODELINE MANAGEMENT

## 4.1 Overview

The Armv8-R AArch64 software stack releases use the following model:

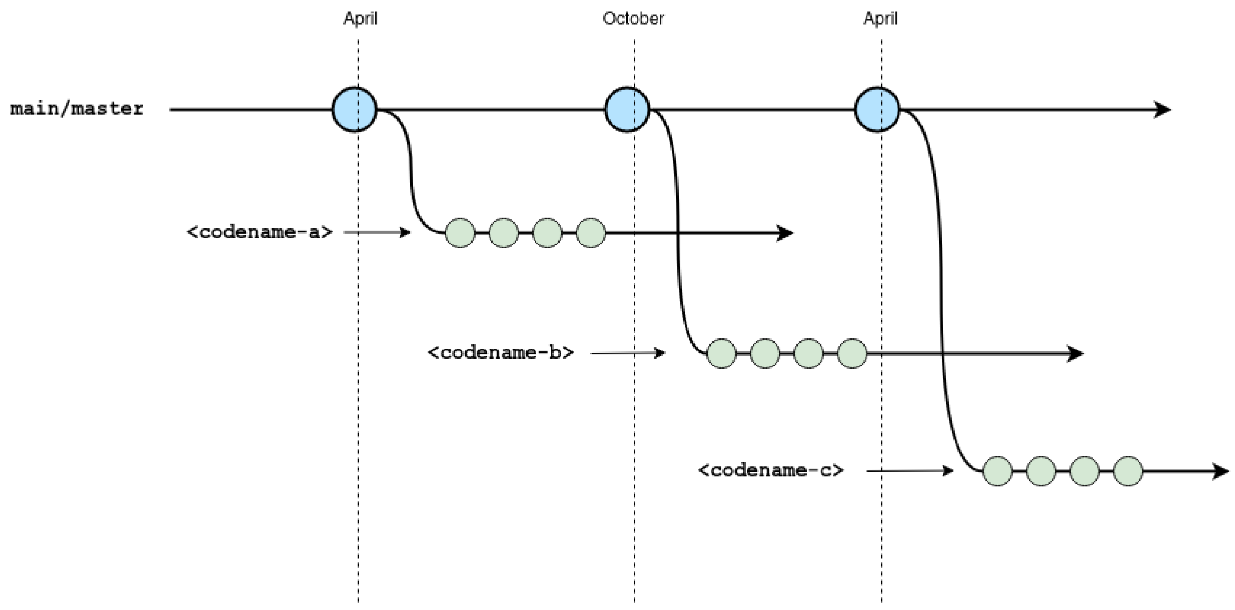
- The software stack releases grow incrementally (meaning that we are not dropping previously released features, with exceptions)

**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



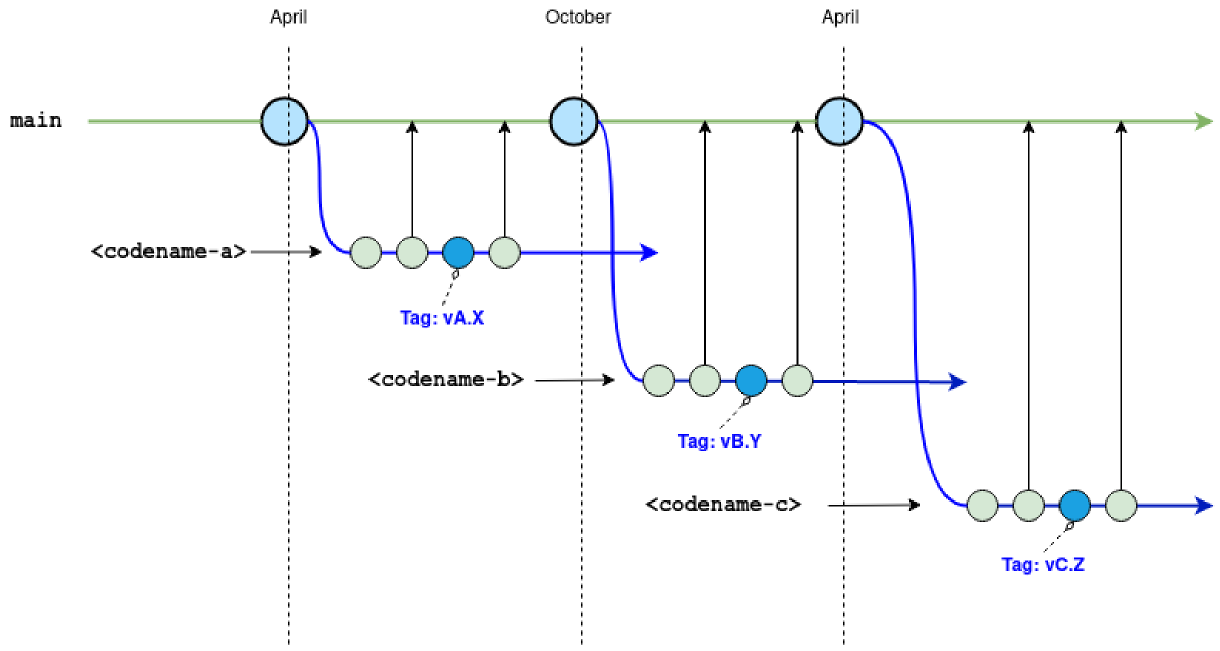
The diagram above gives an overview of the Yocto branch and release process:

- 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).

## 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 4.0 - Xen

### 6.1.1 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

### 6.1.2 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
```

(continues on next page)

(continued from previous page)

```
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
```

### 6.1.3 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*

## 6.2 Release 3 - UEFI

### 6.2.1 Release Note

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

### 6.2.2 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

### 6.2.3 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

### 6.2.4 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*

## 6.3 Release 2 - SMP

### 6.3.1 Release Note

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

### 6.3.2 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

### 6.3.3 Changed

None.

### 6.3.4 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

## 6.4 Release 1 - Single Core

### 6.4.1 Release Note

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

### 6.4.2 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

### 6.4.3 Changed

Initial version.

### 6.4.4 Known Issues and Limitations

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