Shrinkwrap

Release 2026.6.0.dev0

Arm Limited

Apr 02, 2026

CONTENTS

CHAPTER

ONE

CONTENTS

1.1 Overview

1.1.1 Introduction

Shrinkwrap is a tool to simplify the process of building and running firmware on Arm Fixed Virtual Platforms (FVP). It provides a number of configurations that can be used out-of-the-box as well as enabling users to compose and extend them in a manner that can be easily shared and reused. No more shall engineers have to fight with inexplicable fragments of hand-me-down bash code or endlessly search for FVP command line parameters!

Shrinkwrap focuses on building FW stacks and configuring the FVP for a desired set of architecture features so that all components are consistent. Engineers bring their own kernel and rootfs to run on top of this foundation.

Shrinkwrap provides an intuitive command line interface frontend and (by default) a container-based backend so users don't have to think about the tools required to build or run their configs. Everything is also transparent; users can discover every single invoked command with the `--dry-run` option.

Configs are defined in YAML and can easily be composed and extended using the built-in layering system.

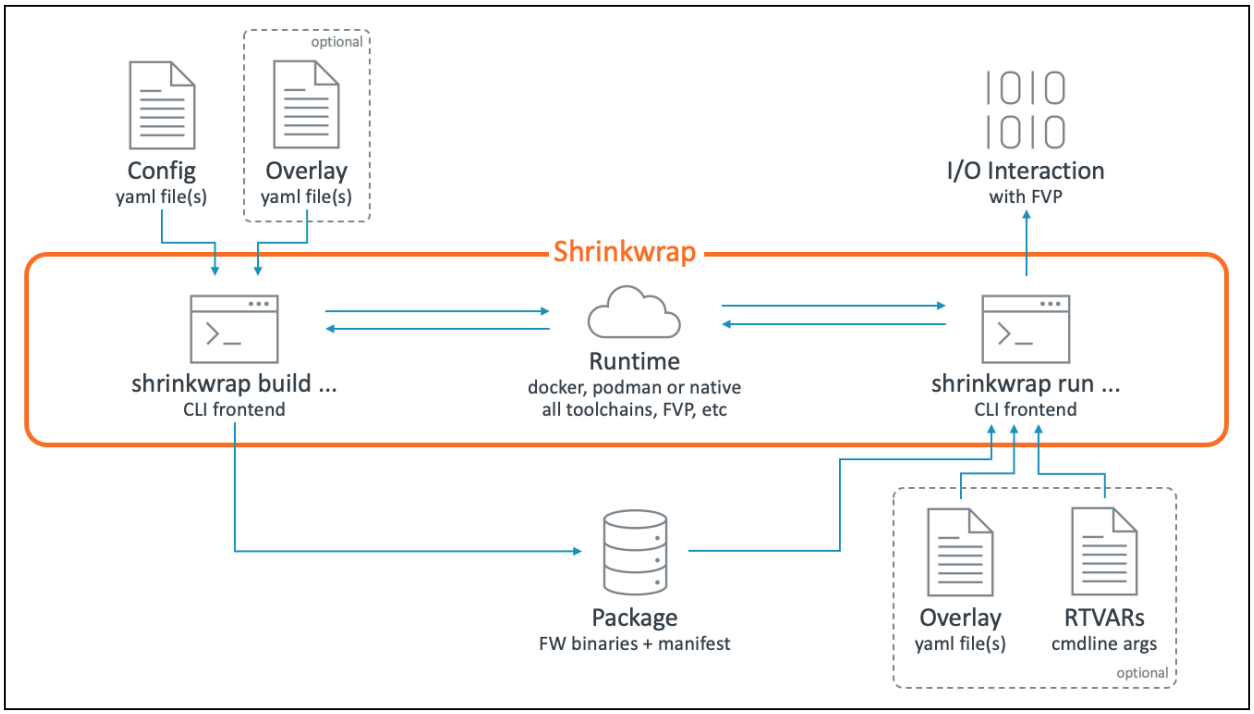
See *Quick Start Guide* to get up and running.

1.1.2 Features

- A simple and intuitive command line interface enables:
  - Acquire and build all required firmware components for a given configuration
  - Package built firmware components for easy distribution
  - Configure and boot the FVP with the packaged firmware components
- Introspect and use any of the supplied the out-of-box configurations
- Create your own configurations by composing with and extending others
- Choose from Docker or Podman runtime backends or run everything natively if you prefer
- Ensure Reproducible builds with supplied runtime container images
- Transparently view the generated bash commands for a given config build or run
- Parallelize builds to make best use of available resources
- Acquire source from Git remote or point to existing Git local repo
- Easily switch between all Arm architecture extensions v8.0 - v9.x.

1.1.3 Architecture

Shrinkwrap is implemented in Python and has a command line interface similar to git, with sub-commands that take options. The Python code parses the supplied config(s) to generate shell commands that are executed in a backend runtime. The runtime is specified by the user and may be `null` (executed natively on the user's system), or a container runtime such as `docker` or `podman`. For the container runtimes, a standard image is provided with all tools preinstalled.



The command line interface has 2 main commands; build and run. The build command takes a configuration (a yaml file) along with an optional overlay which tweaks settings in the config, then builds and packages all the components described within. The package could optionally be distributed for use by others. The run command launches the built package on the FVP, allowing the user to interact with it. The definition of how to run the FVP is contained in the original config that was supplied at build time and is included in the package. The run command allows further tweaking of this runtime config with another optional overlay. The user can also optionally provide values for any runtime variables (RTVARs) that are defined as part of the config. These are typically used to point to the kernel or rootfs that should be used.

1.1.4 Repository Structure

Directory	Description
./config	Shrinkwrap standard config store.
./docker	Scripts to generate docker images used by shrinkwrap's container runtimes.
./documentation	Source for this documentation.
./shrinkwrap	Shrinkwrap Python tool implementation.
./test	Automated tests.

1.1.5 Repository License

The software is provided under an MIT license (more details in *License*).

Contributions to the project should follow the same license.

1.1.6 Contributions and Bug Reports

Contributions are accepted under the MIT license. Only submit contributions where you have authored all of the code.

If you're hitting an error/bug and need help, it's best to raise an issue in GitLab.

1.1.7 Maintainer(s)

- Ryan Roberts <ryan.roberts@arm.com>

1.2 User Guide

1.2.1 Quick Start Guide

Install Shrinkwrap

Packages don't yet exist, so currently the only way to install Shrinkwrap is to install its dependencies and clone the git repository.

Shrinkwrap is tested on **Ubuntu 20.04** although other Linux distributions are likely to JustWork (TM). macOS is also known to work when using the docker runtime as long as Docker Desktop has first been installed.

Shrinkwrap requires **at least Python 3.6.9**. Older versions may work, but are not tested.

```
sudo apt-get install git netcat-openbsd python3 python3-pip telnet
sudo pip3 install pyyaml termcolor tuxmake
git clone https://git.gitlab.arm.com/tooling/shrinkwrap.git
export PATH=$PWD/shrinkwrap/shrinkwrap:$PATH
```

If using a Python version older than 3.9, you will also need to install the `graphlib-backport` pip package:

```
sudo pip3 install graphlib-backport
```

If using Docker Runtime Backend

If Docker was not previously set up on your system, you will need to install the package, create a 'docker' group and add your user to it. This allows shrinkwrap to interact with docker without needing sudo. For more information see docker linux-postinstall.

```
sudo apt-get install docker.io
sudo groupadd docker
sudo usermod -aG docker $USER
# Log out/log in for change to take effect
```

If using Podman Runtime Backend

Note: Podman is only available within Ubuntu repositories from Ubuntu 20.10 and newer. See podman installation instructions for installation methods for other distributions.

```
sudo apt-get install podman
```

Optional Environment Variables

Shrinkwrap consumes the following set of optional environment variables:

name	default	description
SHRINKWRAP_CONFIG	None	Colon-separated list of paths to config stores. Configs are searched for relative to the current directory as well as relative to these paths.
SHRINKWRAP_BUILD	~/.shrinkwrap/build	Path where config builds are performed. Each config has its own subdirectory, with further subdirectories for each of its components.
SHRINKWRAP_PACKAGE	~/.shrinkwrap/package	Path where config builds are packaged to. When running a config, it is done from the package location.
SHRINKWRAP_PROJECT_CACHE	None	Location where cache repositories are stored. This directory contains git trees used as reference when downloading components, allowing to reduce network usage and local storage if the same project is used multiple times. Cache directories ending in ".git" are bare repositories, and ones without the suffix are full repositories. By default no cache is used.

Guided Tour: Configure a platform and boot a kernel

This section provides a guided tour of Shrinkwrap, using a common use case of building required platform FW and configuring the FVP for Armv9.3 and booting a kernel. This example uses EDK2 (UEFI) but many other options are available.

Note: By default, the below commands will use the docker runtime and automatically download and use the appropriate container image from Docker Hub. Alternatively, you can choose to run with the `null` runtime by providing `--runtime=null` (between `shrinkwrap` and the sub-command). This will cause all commands to be executed on the native system. Users are responsible for setting up the environment in this case. Or if you have chosen to use Podman as the runtime backend, add `--runtime=podman`.

First invoke the tool to view help:

```
shrinkwrap --help
shrinkwrap <command> --help
```

Now, inspect the available configs:

```
shrinkwrap inspect
```

This will show all of the (concrete) configs in the config store. The below output shows a sample. Notice that each config lists its runtime variables ("rtvars") along with their default values. `None` means there is no default and the user must provide a value when running the config. (A "concrete" config is one that is deemed ready-to-use out-of-the-box.

Whereas a config "fragment" is a piece of config that is usually composed with others and configured into a concrete config. You can view non-concrete fragments by providing extra args).

```
name:                bootwrapper.yaml

description:         Best choice for: I have a linux-system.axf boot-wrapper and
                     want to run it.

                     This config does not build any components (although
                     shrinkwrap still requires you to build it before running).
                     Instead the user is expected to provide a boot-wrapper
                     executable (usually called linux-system.axf) as the
                     BOOTWRAPPER rtvar, which will be executed in the FVP. A
                     ROOTFS can be optionally provided. If present it is loaded
                     into the virtio block device (/dev/vda).

concrete:            True

run-time variables:  LOCAL_NET_PORT:         8022
                     BOOTWRAPPER:            None
                     ROOTFS:

--------------------------------------------------------------------------------

name:                cca-3world.yaml

description:         Brings together a software stack to demonstrate Arm CCA
                     running on FVP in a three-world configuration. Includes
                     TF-A in root world, RMM in realm world, and Linux in Normal
                     world.

                     In order to launch realm VMs, the user must bring their own
                     rootfs that contains a realm-aware kvmtool and an RSI-aware
                     guest kernel image.

concrete:            True

run-time variables:  LOCAL_NET_PORT:         8022
                     BL1:                    ${artifact:BL1}
                     FIP:                    ${artifact:FIP}
                     KERNEL:                 ${artifact:KERNEL}
                     ROOTFS:

--------------------------------------------------------------------------------

name:                cca-4world.yaml

description:         Brings together a software stack to demonstrate Arm CCA
                     running on FVP in a four-world configuration. Includes TF-A
                     in root world, Hafnium and some demo secure partitions in
                     secure world, RMM in realm world, and Linux in Normal
                     world.
```

(continues on next page)

(continued from previous page)

```
                    In order to launch realm VMs, the user must bring their own
                    rootfs that contains a realm-aware kvmtool and an RSI-aware
                    guest kernel image.

concrete:           True

run-time variables: LOCAL_NET_PORT:         8022
                    BL1:                    ${artifact:BL1}
                    FIP:                    ${artifact:FIP}
                    KERNEL:                 ${artifact:KERNEL}
                    ROOTFS:

--------------------------------------------------------------------------------

name:               ffa-tftf.yaml

description:        Brings together a software stack to demonstrate Arm FF-A
                    running on FVP. Includes TF-A in secure EL3, Hafnium in
                    secure EL2 and some demo TF-A test secure partitions.

concrete:           True

run-time variables: LOCAL_NET_PORT:         8022
                    BL1:                    ${artifact:BL1}
                    FIP:                    ${artifact:FIP}
                    DTB:                    ${artifact:DTB}
                    CMDLINE:                console=ttyAMA0
                                            earlycon=pl011,0x1c090000
                                            root=/dev/vda ip=dhcp
                    KERNEL:                 None
                    ROOTFS:
                    EDK2FLASH:

--------------------------------------------------------------------------------

name:               ns-edk2.yaml

description:        Best choice for: I want to run Linux on FVP, booting with
                    ACPI/DT, and have easy control over its command line.

                    Brings together TF-A and EDK2 to provide a simple non-
                    secure world environment running on FVP. Allows easy
                    specification of the kernel image and command line, and
                    rootfs at runtime (see rtvars). ACPI is provided by UEFI.

                    An extra rtvar is added (DTB) which allows specification of
                    a custom device tree. By default (if not overriding the
                    rtvar), the upstream kernel device tree is used. DT is
                    enabled by default. Use 'acpi=force' to enable ACPI boot.

                    By default (if not overriding the rtvars) a sensible
                    command line is used that will set up the console for
```

(continues on next page)

(continued from previous page)

```
                    logging and attempt to mount the rootfs image from the
                    FVP's virtio block device. However the default rootfs image
                    is empty, so the kernel will panic when attempting to
                    mount; the user must supply a rootfs if it is required that
                    the kernel completes its boot. No default kernel image is
                    supplied and the config will refuse to run unless it is
                    explicitly specified.

                    Note that by default, UEFI variables are build time
                    configured directing EDK2 to boot to the shell. This will
                    cause startup.nsh to be executed and will start the kernel
                    boot. This way everything is automatic. By default, all
                    EDK2 output is muxed to stdout. If you prefer booting UEFI
                    to its UI, override the the build pcd parameter
                    `PcdUefiShellDefaultBootEnable` using the overlay and
                    override terminals 'bp.terminal_0'.type to 'telnet'.
concrete:           True

run-time variables: LOCAL_NET_PORT:         8022
                    BL1:                    ${artifact:BL1}
                    FIP:                    ${artifact:FIP}
                    DTB:                    ${artifact:DTB}
                    CMDLINE:                console=ttyAMA0
                                            earlycon=pl011,0x1c090000
                                            root=/dev/vda ip=dhcp
                    KERNEL:                 None
                    ROOTFS:
                    EDK2FLASH:

--------------------------------------------------------------------------------

name:               ns-preload.yaml

description:        Best choice for: I just want to run Linux on FVP.

                    A simple, non-secure-only configuration where all
                    components are preloaded into memory (TF-A's BL31, DTB and
                    kernel). The system resets directly to BL31. Allows easy
                    specification of a custom command line at build-time (via
                    build.dt.params dictionary) and specification of the device
                    tree, kernel image and rootfs at run-time (see rtvars).

                    By default (if not overriding the rtvars), the upstream
                    kernel device tree is used along with a sensible command
                    line that will set up the console for logging and attempt
                    to mount the rootfs image from the FVP's virtio block
                    device. However the default rootfs image is empty, so the
                    kernel will panic when attempting to mount; the user must
                    supply a rootfs if it is required that the kernel completes
                    its boot. No default kernel image is supplied and the
                    config will refuse to run unless it is explicitly
                    specified.  Note: If specifying a custom dtb at runtime,
```

(continues on next page)

(continued from previous page)

```
                    this will also override any command line specified at build
                    time, since the command line is added to the chosen node of
                    the default dtb.

concrete:           True

run-time variables: LOCAL_NET_PORT:         8022
                    BL31:                   ${artifact:BL31}
                    DTB:                    ${artifact:DTB}
                    KERNEL:                 None
                    ROOTFS:
```

Now build the `ns-edk2.yaml` config. This allows booting a kernel on FVP, using edk2 as the bootloader (it uses DT by default, but can be made to use ACPI by passing `acpi=force` at runtime). (optionally add `--verbose` to see all the output from the component build systems).

```
shrinkwrap build --overlay=arch/v9.3.yaml ns-edk2.yaml
```

This will sync all the required repos, build the components and package the artifacts.

Warning: By default, Shrinkwrap will sync all component repos to the revision specified in the config on every build invocation. If you have made changes in the working directory, shrinkwrap refuses to sync and displays an error. You can override this behaviour so that Shrinkwrap just builds whatever is in the working directory by adding `--no-sync <component>` or `--no-sync-all` to the command line. Alternatively you can force shrinkwrap to override your changes by adding `--force-sync <component>` or `--force-sync-all` to the command line.

Alternatively, pass `--dry-run` to view the shell script that would have been run:

```
shrinkwrap build --overlay=arch/v9.3.yaml --dry-run ns-edk2.yaml
```

```
#!/bin/bash
# SHRINKWRAP AUTOGENERATED SCRIPT.

# Exit on error.
set -e

# Remove old package.
rm -rf package/ns-edk2.yaml > /dev/null 2>&1 || true
rm -rf package/ns-edk2 > /dev/null 2>&1 || true

# Create directory structure.
mkdir -p source/ns-edk2/acpica
mkdir -p source/ns-edk2/dt
mkdir -p source/ns-edk2/edk2
mkdir -p source/ns-edk2/tfa
mkdir -p package/ns-edk2

# Sync git repo for config=ns-edk2 component=acpica.
pushd source/ns-edk2
if [ ! -e "acpica/.git" ] || [ -f "./.acpica_sync" ]; then
      rm -rf acpica > /dev/null 2>&1 || true
```

(continues on next page)

(continued from previous page)

```
      mkdir -p .
      touch ./.acpica_sync
      git clone --quiet https://github.com/acpica/acpica.git acpica
      pushd acpica
      git checkout --quiet --force R06_28_23
      git submodule --quiet update --init --checkout --recursive --force
      popd
      rm ./.acpica_sync
else
      pushd acpica
      if ! git checkout --quiet  R06_28_23 > /dev/null 2>&1 &&
         ! ( git remote set-url origin https://github.com/acpica/acpica.git &&
             git fetch --quiet --prune --tags origin &&
             git checkout --quiet  R06_28_23) ||
         ! git submodule --quiet update --init --checkout --recursive
      then
              echo "note: use --force-sync=acpica to override any change"
              exit 1
      fi
      popd
fi
popd

# Sync git repo for config=ns-edk2 component=dt.
pushd source/ns-edk2
if [ ! -e "dt/.git" ] || [ -f "./.dt_sync" ]; then
      rm -rf dt > /dev/null 2>&1 || true
      mkdir -p .
      touch ./.dt_sync
      git clone --quiet https://git.kernel.org/pub/scm/linux/kernel/git/devicetree/
↪devicetree-rebasing.git dt
      pushd dt
      git checkout --quiet --force v6.6-dts
      git submodule --quiet update --init --checkout --recursive --force
      popd
      rm ./.dt_sync
else
      pushd dt
      if ! git checkout --quiet  v6.6-dts > /dev/null 2>&1 &&
         ! ( git remote set-url origin https://git.kernel.org/pub/scm/linux/kernel/git/
↪devicetree/devicetree-rebasing.git &&
             git fetch --quiet --prune --tags origin &&
             git checkout --quiet  v6.6-dts) ||
         ! git submodule --quiet update --init --checkout --recursive
      then
              echo "note: use --force-sync=dt to override any change"
              exit 1
      fi
      popd
fi
popd
```

(continues on next page)

(continued from previous page)

```
# Sync git repo for config=ns-edk2 component=edk2.
pushd source/ns-edk2
if [ ! -e "edk2/edk2/.git" ] || [ -f "edk2/.edk2_sync" ]; then
      rm -rf edk2/edk2 > /dev/null 2>&1 || true
      mkdir -p edk2
      touch edk2/.edk2_sync
      git clone --quiet https://github.com/tianocore/edk2.git edk2/edk2
      pushd edk2/edk2
      git checkout --quiet --force edk2-stable202311
      git submodule --quiet update --init --checkout --recursive --force
      popd
      rm edk2/.edk2_sync
else
      pushd edk2/edk2
      if ! git checkout --quiet  edk2-stable202311 > /dev/null 2>&1 &&
         ! ( git remote set-url origin https://github.com/tianocore/edk2.git &&
             git fetch --quiet --prune --tags origin &&
             git checkout --quiet  edk2-stable202311) ||
         ! git submodule --quiet update --init --checkout --recursive
      then
              echo "note: use --force-sync=edk2 to override any change"
              exit 1
      fi
      popd
fi
if [ ! -e "edk2/edk2-platforms/.git" ] || [ -f "edk2/.edk2-platforms_sync" ]; then
      rm -rf edk2/edk2-platforms > /dev/null 2>&1 || true
      mkdir -p edk2
      touch edk2/.edk2-platforms_sync
      git clone --quiet https://github.com/tianocore/edk2-platforms.git edk2/edk2-
↪platforms
      pushd edk2/edk2-platforms
      git checkout --quiet --force 4b07df2e6f3813c6e955197dacb2cdfbe3471caa
      git submodule --quiet update --init --checkout --recursive --force
      popd
      rm edk2/.edk2-platforms_sync
else
      pushd edk2/edk2-platforms
      if ! git checkout --quiet  4b07df2e6f3813c6e955197dacb2cdfbe3471caa > /dev/null 2>&
↪1 &&
         ! ( git remote set-url origin https://github.com/tianocore/edk2-platforms.git &&
             git fetch --quiet --prune --tags origin &&
             git checkout --quiet  4b07df2e6f3813c6e955197dacb2cdfbe3471caa) ||
         ! git submodule --quiet update --init --checkout --recursive
      then
              echo "note: use --force-sync=edk2 to override any change"
              exit 1
      fi
      popd
fi
popd
```

(continues on next page)

(continued from previous page)

```
# Sync git repo for config=ns-edk2 component=tfa.
pushd source/ns-edk2
if [ ! -e "tfa/.git" ] || [ -f "./.tfa_sync" ]; then
      rm -rf tfa > /dev/null 2>&1 || true
      mkdir -p .
      touch ./.tfa_sync
      git clone --quiet https://git.trustedfirmware.org/TF-A/trusted-firmware-a.git tfa
      pushd tfa
      git checkout --quiet --force v2.11
      git submodule --quiet update --init --checkout --recursive --force
      popd
      rm ./.tfa_sync
else
      pushd tfa
      if ! git checkout --quiet  v2.11 > /dev/null 2>&1 &&
         ! ( git remote set-url origin https://git.trustedfirmware.org/TF-A/trusted-
↪firmware-a.git &&
             git fetch --quiet --prune --tags origin &&
             git checkout --quiet  v2.11) ||
         ! git submodule --quiet update --init --checkout --recursive
      then
              echo "note: use --force-sync=tfa to override any change"
              exit 1
      fi
      popd
fi
popd

# Build for config=ns-edk2 component=acpica.
export CROSS_COMPILE=
pushd source/ns-edk2/acpica
rm -rf source/ns-edk2/acpica/generate/unix/acpica
make -j4
mv source/ns-edk2/acpica/generate/unix/bin source/ns-edk2/acpica/generate/unix/acpica
popd

# Build for config=ns-edk2 component=dt.
export CROSS_COMPILE=aarch64-none-elf-
pushd source/ns-edk2/dt
DTS=fvp-base-revc.dts
INITRD_START=
INITRD_END=
DT_BASENAME=$(basename ${DTS} .dts)
DTB_INTER=src/arm64/arm/${DT_BASENAME}.dtb
DTB_FINAL=build/ns-edk2/dt/dt_bootargs.dtb
make CPP=${CROSS_COMPILE}cpp -j4 ${DTB_INTER}
CHOSEN=
if [ ! -z "" ]; then
CHOSEN="${CHOSEN}bootargs = \"\";\n"
fi
if [ ! -z "${INITRD_START}" ] && [ ! -z "${INITRD_END}" ]; then
INITRD_START_HI=$((${INITRD_START} >> 32) & 0xffffffff))
```

(continues on next page)

(continued from previous page)

```
INITRD_START_LO=$((${INITRD_START} & 0xffffffff))
INITRD_END_HI=$((${INITRD_END} >> 32) & 0xffffffff))
INITRD_END_LO=$((${INITRD_END} & 0xffffffff))
CHOSEN="${CHOSEN}linux,initrd-start = <${INITRD_START_HI} ${INITRD_START_LO}>;\n"
CHOSEN="${CHOSEN}linux,initrd-end = <${INITRD_END_HI} ${INITRD_END_LO}>;\n"
fi
if [ -z "${CHOSEN}" ]; then
cp ${DTB_INTER} ${DTB_FINAL}
else
( dtc -q -O dts -I dtb ${DTB_INTER} ; echo -e "/ { chosen { ${CHOSEN} }; };" ) | dtc -q -
↪O dtb -o ${DTB_FINAL}
fi
if [ "${DTS}" = "fvp-base-revc.dts" ]; then
OVERLAY="/ {
  reserved-memory {
    fw: fw@7C000000 {
      reg = <0x00000000 0xFC000000 0 0x04000000>;
      no-map;
    };
  };
  timer {
    clock-frequency = <100000000>;
  };
  psci {
    compatible = \"arm,psci-1.0\", \"arm,psci-0.2\";
    max-pwr-lvl = <2>;
  };
  cpus {
    cpu-map {
      cluster0 {
        core0 { cpu = <&{/cpus/cpu@0}>; };
        core1 { cpu = <&{/cpus/cpu@100}>; };
        core2 { cpu = <&{/cpus/cpu@200}>; };
        core3 { cpu = <&{/cpus/cpu@300}>; };
      };
      cluster1 {
        core0 { cpu = <&{/cpus/cpu@10000}>; };
        core1 { cpu = <&{/cpus/cpu@10100}>; };
        core2 { cpu = <&{/cpus/cpu@10200}>; };
        core3 { cpu = <&{/cpus/cpu@10300}>; };
      };
    };
  };
  bus@8000000 {
    motherboard-bus@8000000 {
      iofpga-bus@300000000 {
        virtio@200000 {
          status = \"okay\";
        };
      };
    };
  };
```

(continues on next page)

(continued from previous page)

```
};"
( dtc -q -O dts -I dtb ${DTB_FINAL} ; echo -e "${OVERLAY}" ) | dtc -q -O dtb -o ${DTB_
↪FINAL}
fi
popd

# Copy artifacts for config=ns-edk2 component=acpica.
cp -r source/ns-edk2/acpica/generate/unix/acpica package/ns-edk2/acpica

# Build for config=ns-edk2 component=edk2.
export CROSS_COMPILE=aarch64-none-elf-
pushd source/ns-edk2/edk2
export WORKSPACE=source/ns-edk2/edk2
export GCC_AARCH64_PREFIX=$CROSS_COMPILE
export PACKAGES_PATH=$WORKSPACE/edk2:$WORKSPACE/edk2-platforms
export IASL_PREFIX=source/ns-edk2/acpica/generate/unix/acpica/
export PYTHON_COMMAND=/usr/bin/python3
source edk2/edksetup.sh --reconfig
make -j4 -C edk2/BaseTools
build -n 4 -D EDK2_OUT_DIR=build/ns-edk2/edk2 -a AARCH64 -t GCC -p Platform/ARM/
↪VExpressPkg/ArmVExpress-FVP-AArch64.dsc -b RELEASE --pcd PcdShellDefaultDelay=0  --pcd␣
↪PcdUefiShellDefaultBootEnable=1
popd

# Copy artifacts for config=ns-edk2 component=dt.
cp -r build/ns-edk2/dt/dt_bootargs.dtb package/ns-edk2/dt_bootargs.dtb

# Copy artifacts for config=ns-edk2 component=edk2.
cp -r build/ns-edk2/edk2/RELEASE_GCC/FV/FVP_AARCH64_EFI.fd package/ns-edk2/FVP_AARCH64_
↪EFI.fd

# Build for config=ns-edk2 component=tfa.
export CROSS_COMPILE=aarch64-none-elf-
pushd source/ns-edk2/tfa
make BUILD_BASE=build/ns-edk2/tfa LOG_LEVEL=40 ARM_ARCH_MINOR=2 DEBUG=0 ARM_DISABLE_
↪TRUSTED_WDOG=1 CTX_INCLUDE_AARCH32_REGS=0 BRANCH_PROTECTION=1 ARM_ARCH_MAJOR=9␣
↪PLAT=fvp BL33=build/ns-edk2/edk2/RELEASE_GCC/FV/FVP_AARCH64_EFI.fd FVP_HW_CONFIG_
↪DTS=fdts/fvp-base-gicv3-psci-1t.dts -j$(( 4 < 8 ? 4 : 8 )) all fip
popd

# Copy artifacts for config=ns-edk2 component=tfa.
cp -r build/ns-edk2/tfa/fvp/release/bl31.bin package/ns-edk2/bl31.bin
cp -r build/ns-edk2/tfa/fvp/release/bl1.bin package/ns-edk2/bl1.bin
cp -r build/ns-edk2/tfa/fvp/release/fip.bin package/ns-edk2/fip.bin
cp -r build/ns-edk2/tfa/fvp/release/bl2.bin package/ns-edk2/bl2.bin
```

Now start the FVP. We will pass our own kernel and rootfs disk image as runtime variables. A config can define any number of runtime variables which may have default values (see `inspect` command above). If a variable has no default value, then the user must provide a value when invoking the `run` command. The `ns-edk2.yaml` config requires the user to provide a kernel, but the rootfs is optional. If the rootfs was omitted, the kernel would boot to the point where it attempts to mount the rootfs then panic (which is sufficient for some development use cases!).

```
shrinkwrap run --rtvar=KERNEL=path/to/Image --rtvar=ROOTFS=path/to/rootfs.img ns-edk2.
↪yaml
```

This starts the FVP and multiplexes all the UART terminals to stdout and forwards stdin to the `tfa+linux` uart terminal. This allows the user to interact directly with the FVP in a terminal without the need for a GUI setup:

```
[       fvp ] terminal_0: Listening for serial connection on port 5000
[       fvp ] terminal_1: Listening for serial connection on port 5001
[       fvp ] terminal_2: Listening for serial connection on port 5002
[       fvp ] terminal_3: Listening for serial connection on port 5003
[       fvp ]
[       fvp ] Info: FVP_Base_RevC_2xAEMvA: FVP_Base_RevC_2xAEMvA.bp.flashloader0:␣
↪FlashLoader: Loaded 100 kB from file '<root>/package/ns-preload/fip.bin'
[       fvp ]
[       fvp ] Info: FVP_Base_RevC_2xAEMvA: FVP_Base_RevC_2xAEMvA.bp.secureflashloader:␣
↪FlashLoader: Loaded 30 kB from file '<root>/package/ns-preload/bl1.bin'
[       fvp ]
[       fvp ] libdbus-1.so.3: cannot open shared object file: No such file or directory
[       fvp ] libdbus-1.so.3: cannot open shared object file: No such file or directory
[ tfa+linux ] NOTICE:  BL31: v2.7(release):v2.7.0-391-g9dedc1ab2
[ tfa+linux ] NOTICE:  BL31: Built : 09:41:20, Sep 15 2022
[ tfa+linux ] INFO:    GICv3 with legacy support detected.
[ tfa+linux ] INFO:    ARM GICv3 driver initialized in EL3
[ tfa+linux ] INFO:    Maximum SPI INTID supported: 255
[ tfa+linux ] INFO:    Configuring TrustZone Controller
[ tfa+linux ] INFO:    Total 8 regions set.
[ tfa+linux ] INFO:    BL31: Initializing runtime services
[ tfa+linux ] INFO:    BL31: Preparing for EL3 exit to normal world
[ tfa+linux ] INFO:    Entry point address = 0x84000000
[ tfa+linux ] INFO:    SPSR = 0x3c9
[ tfa+linux ] [    0.000000] Booting Linux on physical CPU 0x0000000000 [0x410fd0f0]
[ tfa+linux ] [    0.000000] Linux version 5.15.0-rc2-gca9bfbea162d (ryarob01@e125769)␣
↪(aarch64-none-linux-gnu-gcc (GNU Toolchain for the A-profile Architecture 9.2-2019.12␣
↪(arm-9.10)) 9.2.1 20191025, GNU ld (GNU Toolchain for the A-profile Architecture 9.2-
↪2019.12 (arm-9.10)) 2.33.1.20191209) #1 SMP PREEMPT Thu Aug 4 11:31:55 BST 2022
[ tfa+linux ] [    0.000000] Machine model: FVP Base RevC
[ tfa+linux ] [    0.000000] earlycon: pl11 at MMIO 0x000000001c090000 (options '')
[ tfa+linux ] [    0.000000] printk: bootconsole [pl11] enabled
[ tfa+linux ] [    0.000000] efi: UEFI not found.
[ tfa+linux ] [    0.000000] Reserved memory: created DMA memory pool at␣
↪0x0000000018000000, size 8 MiB
[ tfa+linux ] [    0.000000] OF: reserved mem: initialized node vram@18000000,␣
↪compatible id shared-dma-pool
[ tfa+linux ] [    0.000000] NUMA: No NUMA configuration found
[ tfa+linux ] [    0.000000] NUMA: Faking a node at [mem 0x0000000080000000-
↪0x00000008fffffff]
[ tfa+linux ] [    0.000000] NUMA: NODE_DATA [mem 0x8ff7efc00-0x8ff7f1fff]
[ tfa+linux ] [    0.000000] Zone ranges:
[ tfa+linux ] [    0.000000]   DMA      [mem 0x0000000080000000-0x00000000ffffffff]
[ tfa+linux ] [    0.000000]   DMA32    empty
[ tfa+linux ] [    0.000000]   Normal   [mem 0x0000000100000000-0x00000008fffffff]
[ tfa+linux ] [    0.000000] Movable zone start for each node
[ tfa+linux ] [    0.000000] Early memory node ranges
```

(continues on next page)

(continued from previous page)

```
[ tfa+linux ] [    0.000000]   node   0: [mem 0x0000000080000000-0x00000000ffffffff]
[ tfa+linux ] [    0.000000]   node   0: [mem 0x0000000880000000-0x00000008ffffffff]
[ tfa+linux ] [    0.000000] Initmem setup node 0 [mem 0x0000000080000000-
↪0x00000008ffffffff]
[ tfa+linux ] [    0.000000] cma: Reserved 32 MiB at 0x00000000fe000000
[ tfa+linux ] [    0.000000] psci: probing for conduit method from DT.
[ tfa+linux ] [    0.000000] psci: PSCIv1.1 detected in firmware.
[ tfa+linux ] [    0.000000] psci: Using standard PSCI v0.2 function IDs
[ tfa+linux ] [    0.000000] psci: MIGRATE_INFO_TYPE not supported.
[ tfa+linux ] [    0.000000] psci: SMC Calling Convention v1.2
...
```

Alternatively, you could have passed `--dry-run` to see the FVP invocation script:

```
shrinkwrap run --rtvar=KERNEL=path/to/Image --rtvar=ROOTFS=path/to/rootfs.img --dry-run␣
↪ns-edk2.yaml
```

```
#!/bin/bash
# SHRINKWRAP AUTOGENERATED SCRIPT.

# Exit on error.
set -e

# Execute prerun commands.
SEMIHOSTDIR=`mktemp -d`
function finish { rm -rf $SEMIHOSTDIR; }
trap finish EXIT
cp ./path/to/Image ${SEMIHOSTDIR}/Image
cp <root>/package/ns-edk2/dt_bootargs.dtb ${SEMIHOSTDIR}/fdt.dtb
cat <<EOF > ${SEMIHOSTDIR}/startup.nsh
Image dtb=fdt.dtb console=ttyAMA0 earlycon=pl011,0x1c090000 root=/dev/vda ip=dhcp
EOF

# Run the model.
FVP_Base_RevC-2xAEMvA \
    --stat \
    -C bp.dram_size=4 \
    -C bp.flashloader0.fname=<root>/package/ns-edk2/fip.bin \
    -C bp.flashloader1.fname= \
    -C bp.hostbridge.userNetPorts=8022=22 \
    -C bp.hostbridge.userNetworking=1 \
    -C bp.refcounter.non_arch_start_at_default=1 \
    -C bp.secure_memory=1 \
    -C bp.secureflashloader.fname=<root>/package/ns-edk2/bl1.bin \
    -C bp.smsc_91c111.enabled=1 \
    -C bp.terminal_0.mode=telnet \
    -C bp.terminal_0.start_telnet=0 \
    -C bp.terminal_1.mode=raw \
    -C bp.terminal_1.start_telnet=0 \
    -C bp.terminal_2.mode=raw \
    -C bp.terminal_2.start_telnet=0 \
    -C bp.terminal_3.mode=raw \
```

(continues on next page)

(continued from previous page)

```
    -C bp.terminal_3.start_telnet=0 \
    -C bp.ve_sysregs.exit_on_shutdown=1 \
    -C bp.virtio_rng.enabled=1 \
    -C bp.virtioblockdevice.image_path=./path/to/rootfs.img \
    -C bp.virtiop9device.root_path= \
    -C bp.vis.disable_visualisation=1 \
    -C cache_state_modelled=0 \
    -C cluster0.NUM_CORES=4 \
    -C cluster0.PA_SIZE=48 \
    -C cluster0.check_memory_attributes=0 \
    -C cluster0.clear_reg_top_eret=2 \
    -C cluster0.cpu0.semihosting-cwd=${SEMIHOSTDIR} \
    -C cluster0.ecv_support_level=2 \
    -C cluster0.enhanced_pac2_level=3 \
    -C cluster0.gicv3.cpuintf-mmap-access-level=2 \
    -C cluster0.gicv3.without-DS-support=1 \
    -C cluster0.gicv4.mask-virtual-interrupt=1 \
    -C cluster0.has_16k_granule=1 \
    -C cluster0.has_amu=1 \
    -C cluster0.has_arm_v8-1=1 \
    -C cluster0.has_arm_v8-2=1 \
    -C cluster0.has_arm_v8-3=1 \
    -C cluster0.has_arm_v8-4=1 \
    -C cluster0.has_arm_v8-5=1 \
    -C cluster0.has_arm_v8-6=1 \
    -C cluster0.has_arm_v8-7=1 \
    -C cluster0.has_arm_v8-8=1 \
    -C cluster0.has_arm_v9-0=1 \
    -C cluster0.has_arm_v9-1=1 \
    -C cluster0.has_arm_v9-2=1 \
    -C cluster0.has_arm_v9-3=1 \
    -C cluster0.has_branch_target_exception=1 \
    -C cluster0.has_brbe=1 \
    -C cluster0.has_brbe_v1p1=1 \
    -C cluster0.has_const_pac=1 \
    -C cluster0.has_hpmn0=1 \
    -C cluster0.has_large_system_ext=1 \
    -C cluster0.has_large_va=1 \
    -C cluster0.has_rndr=1 \
    -C cluster0.has_sve=1 \
    -C cluster0.max_32bit_el=0 \
    -C cluster0.pmb_idr_external_abort=1 \
    -C cluster0.stage12_tlb_size=1024 \
    -C cluster0.sve.has_sme2=1 \
    -C cluster0.sve.has_sme=1 \
    -C cluster0.sve.has_sve2=1 \
    -C cluster1.NUM_CORES=4 \
    -C cluster1.PA_SIZE=48 \
    -C cluster1.check_memory_attributes=0 \
    -C cluster1.clear_reg_top_eret=2 \
    -C cluster1.ecv_support_level=2 \
    -C cluster1.enhanced_pac2_level=3 \
```

(continues on next page)

(continued from previous page)

```
    -C cluster1.gicv3.cpuintf-mmap-access-level=2 \
    -C cluster1.gicv3.without-DS-support=1 \
    -C cluster1.gicv4.mask-virtual-interrupt=1 \
    -C cluster1.has_16k_granule=1 \
    -C cluster1.has_amu=1 \
    -C cluster1.has_arm_v8-1=1 \
    -C cluster1.has_arm_v8-2=1 \
    -C cluster1.has_arm_v8-3=1 \
    -C cluster1.has_arm_v8-4=1 \
    -C cluster1.has_arm_v8-5=1 \
    -C cluster1.has_arm_v8-6=1 \
    -C cluster1.has_arm_v8-7=1 \
    -C cluster1.has_arm_v8-8=1 \
    -C cluster1.has_arm_v9-0=1 \
    -C cluster1.has_arm_v9-1=1 \
    -C cluster1.has_arm_v9-2=1 \
    -C cluster1.has_arm_v9-3=1 \
    -C cluster1.has_branch_target_exception=1 \
    -C cluster1.has_brbe=1 \
    -C cluster1.has_brbe_v1p1=1 \
    -C cluster1.has_const_pac=1 \
    -C cluster1.has_hpmn0=1 \
    -C cluster1.has_large_system_ext=1 \
    -C cluster1.has_large_va=1 \
    -C cluster1.has_rndr=1 \
    -C cluster1.has_sve=1 \
    -C cluster1.max_32bit_el=0 \
    -C cluster1.pmb_idr_external_abort=1 \
    -C cluster1.stage12_tlb_size=1024 \
    -C cluster1.sve.has_sme2=1 \
    -C cluster1.sve.has_sme=1 \
    -C cluster1.sve.has_sve2=1 \
    -C gic_distributor.has_nmi=1 \
    -C pci.pci_smmuv3.mmu.SMMU_AIDR=2 \
    -C pci.pci_smmuv3.mmu.SMMU_IDR0=135263935 \
    -C pci.pci_smmuv3.mmu.SMMU_IDR1=216481056 \
    -C pci.pci_smmuv3.mmu.SMMU_IDR3=5908 \
    -C pci.pci_smmuv3.mmu.SMMU_IDR5=4294902901 \
    -C pci.pci_smmuv3.mmu.SMMU_S_IDR1=2684354562 \
    -C pci.pci_smmuv3.mmu.SMMU_S_IDR2=0 \
    -C pci.pci_smmuv3.mmu.SMMU_S_IDR3=0 \
    -C pctl.startup=0.0.0.0
```

Overlays are an important concept for Shrinkwrap. An overlay is a config fragment (either a yaml file or a json-encoded string) that can be passed separately on the command line and forms the top layer of the config. In this way, it can override or add any required configuration. You could achieve the same effect by creating a new config and specifying the main config as a layer in that new config, but with an overlay, you can apply a config fragment to many different existing configs without the need to write a new config file each time. You can see overlays being using in the above commands to target a specific Arm architecture revision (v9.3 in the example). You can change the targeted architecture just by changing the overlay. There are many other places where overlays come in handy. See *Shrinkwrap Recipes* for more examples.

You will notice in the examples above, that only `build` commands include the overlay and `run` commands don't specify

it. This is because the final config used for building is packaged in the built package, so when running the package, the presence of the overlay is implicit. However, a user could choose to provide an extra overlay at `run` time, that affects only the runtime portion to customize even further if desired.

For debug purposes, you can see a final, merged config by using the `process` command:

```
shrinkwrap process --action=merge --overlay=arch/v9.3.yaml ns-edk2.yaml
```

```
%YAML 1.2
---
name: ns-edk2
fullname: ns-edk2.yaml
description: "Best choice for: I want to run Linux on FVP, booting with ACPI/DT, and\
  \ have easy control over its command line.\nBrings together TF-A and EDK2 to provide\
  \ a simple non-secure world environment running on FVP. Allows easy specification\
  \ of the kernel image and command line, and rootfs at runtime (see rtvars). ACPI\
  \ is provided by UEFI.\nAn extra rtvar is added (DTB) which allows specification\
  \ of a custom device tree. By default (if not overriding the rtvar), the upstream\
  \ kernel device tree is used. DT is enabled by default. Use 'acpi=force' to enable\
  \ ACPI boot.\nBy default (if not overriding the rtvars) a sensible command line\
  \ is used that will set up the console for logging and attempt to mount the rootfs\
  \ image from the FVP's virtio block device. However the default rootfs image is\
  \ empty, so the kernel will panic when attempting to mount; the user must supply\
  \ a rootfs if it is required that the kernel completes its boot. No default kernel\
  \ image is supplied and the config will refuse to run unless it is explicitly␣
↪specified.\n\
  Note that by default, UEFI variables are build time configured directing EDK2 to\
  \ boot to the shell. This will cause startup.nsh to be executed and will start the\
  \ kernel boot. This way everything is automatic. By default, all EDK2 output is\
  \ muxed to stdout. If you prefer booting UEFI to its UI, override the the build\
  \ pcd parameter `PcdUefiShellDefaultBootEnable` using the overlay and override␣
↪terminals\
  \ 'bp.terminal_0'.type to 'telnet'.\nWhen booting with device tree, a directory\
  \ can optionally be shared from the host system into the Linux environment running\
  \ in the FVP. To do so, set the SHARE rtvar to the desired directory, then mount\
  \ the share inside the FVP with the following (or automate it in fstab):\n.. code-
↪block::\
  \ shell\n  # mkdir /share\n  # mount -t 9p -o trans=virtio,version=9p2000.L FM /share"
image: null
concrete: true
graph: {}
build:
  acpica:
    repo:
      .:
        remote: https://github.com/acpica/acpica.git
        revision: R06_28_23
    sync: null
    sourcedir: null
    builddir: null
    toolchain: null
    stderrfilt: null
    params: {}
    prebuild: []
```

(continues on next page)

(continued from previous page)

```
    build:
    - rm -rf ${param:sourcedir}/generate/unix/acpica
    - make -j${param:jobs}
    - mv ${param:sourcedir}/generate/unix/bin ${param:sourcedir}/generate/unix/acpica
    postbuild: []
    artifacts:
      ACPICA: ${param:sourcedir}/generate/unix/acpica
  dt:
    repo:
      .:
        remote: https://git.kernel.org/pub/scm/linux/kernel/git/devicetree/devicetree-
↪rebasing.git
        revision: v6.6-dts
    sync: null
    sourcedir: null
    builddir: null
    toolchain: aarch64-none-elf-
    stderrfilt: null
    params: {}
    prebuild:
    - DTS=fvp-base-revc.dts
    - INITRD_START=
    - INITRD_END=
    build:
    - DT_BASENAME=$$(basename $${DTS} .dts)
    - DTB_INTER=src/arm64/arm/$${DT_BASENAME}.dtb
    - DTB_FINAL=${param:builddir}/dt_bootargs.dtb
    - make CPP=$${CROSS_COMPILE}cpp -j${param:jobs} $${DTB_INTER}
    - CHOSEN=
    - if [ ! -z "${param:join_equal}" ]; then
    - CHOSEN="$${CHOSEN}bootargs = \"${param:join_equal}\";\n"
    - fi
    - if [ ! -z "$${INITRD_START}" ] && [ ! -z "$${INITRD_END}" ]; then
    - INITRD_START_HI=$$((($${INITRD_START} >> 32) & 0xffffffff))
    - INITRD_START_LO=$$(($${INITRD_START} & 0xffffffff))
    - INITRD_END_HI=$$((($${INITRD_END} >> 32) & 0xffffffff))
    - INITRD_END_LO=$$(($${INITRD_END} & 0xffffffff))
    - CHOSEN="$${CHOSEN}linux,initrd-start = <$${INITRD_START_HI} $${INITRD_START_LO}>;\n
↪"
    - CHOSEN="$${CHOSEN}linux,initrd-end = <$${INITRD_END_HI} $${INITRD_END_LO}>;\n"
    - fi
    - if [ -z "$${CHOSEN}" ]; then
    - cp $${DTB_INTER} $${DTB_FINAL}
    - else
    - ( dtc -q -O dts -I dtb $${DTB_INTER} ; echo -e "/ { chosen { $${CHOSEN} }; };"
      ) | dtc -q -O dtb -o $${DTB_FINAL}
    - fi
    - if [ "$${DTS}" = "fvp-base-revc.dts" ]; then
    - "OVERLAY=\"/ {\n  reserved-memory {\n    fw: fw@7C000000 {\n      reg =
↪<0x00000000\
      \ 0xFC000000 0 0x04000000>;\n      no-map;\n    };\n  };\n  timer {\n    clock-
↪frequency\
```

(continues on next page)

(continued from previous page)

```
      \ = <100000000>;\n  };\n  psci {\n    compatible = \\\"arm,psci-1.0\\\", \\\"\
      arm,psci-0.2\\\";\n    max-pwr-lvl = <2>;\n  };\n  cpus {\n    cpu-map {\n \
      \     cluster0 {\n        core0 { cpu = <&{/cpus/cpu@0}>; };\n        core1\
      \ { cpu = <&{/cpus/cpu@100}>; };\n        core2 { cpu = <&{/cpus/cpu@200}>;\
      \ };\n        core3 { cpu = <&{/cpus/cpu@300}>; };\n      };\n      cluster1\
      \ {\n        core0 { cpu = <&{/cpus/cpu@10000}>; };\n        core1 { cpu = <&{/
↪cpus/cpu@10100}>;\
      \ };\n        core2 { cpu = <&{/cpus/cpu@10200}>; };\n        core3 { cpu =\
      \ <&{/cpus/cpu@10300}>; };\n      };\n    };\n  };\n  bus@8000000 {\n    ␣
↪motherboard-bus@8000000\
      \ {\n      iofpga-bus@300000000 {\n        virtio@200000 {\n          status\
      \ = \\\"okay\\\";\n        };\n      };\n    };\n  };\n};\""
    - ( dtc -q -O dts -I dtb ${DTB_FINAL} ; echo -e "${OVERLAY}" ) | dtc -q -O dtb
      -o ${DTB_FINAL}
    - fi
    postbuild: []
    artifacts:
      DTB: ${param:builddir}/dt_bootargs.dtb
  edk2:
    repo:
      edk2:
        remote: https://github.com/tianocore/edk2.git
        revision: edk2-stable202311
      edk2-platforms:
        remote: https://github.com/tianocore/edk2-platforms.git
        revision: 4b07df2e6f3813c6e955197dacb2cdfbe3471caa
    sync: null
    sourcedir: null
    builddir: null
    toolchain: aarch64-none-elf-
    stderrfilt: true
    params:
      -a: AARCH64
      -t: GCC
      -p: Platform/ARM/VExpressPkg/ArmVExpress-FVP-AArch64.dsc
      -b: RELEASE
      --pcd: PcdShellDefaultDelay=0
      ' --pcd': PcdUefiShellDefaultBootEnable=1
    prebuild:
    - export WORKSPACE=${param:sourcedir}
    - export GCC_AARCH64_PREFIX=$$CROSS_COMPILE
    - export PACKAGES_PATH=$$WORKSPACE/edk2:$$WORKSPACE/edk2-platforms
    - export IASL_PREFIX=${artifact:ACPICA}/
    - export PYTHON_COMMAND=/usr/bin/python3
    build:
    - source edk2/edksetup.sh --reconfig
    - make -j${param:jobs} -C edk2/BaseTools
    - build -n ${param:jobs} -D EDK2_OUT_DIR=${param:builddir} ${param:join_space}
    postbuild: []
    artifacts:
      EDK2: ${param:builddir}/RELEASE_GCC/FV/FVP_AARCH64_EFI.fd
  tfa:
```

(continues on next page)

(continued from previous page)

```
    repo:
      .:
        remote: https://git.trustedfirmware.org/TF-A/trusted-firmware-a.git
        revision: v2.11
    sync: null
    sourcedir: null
    builddir: null
    toolchain: aarch64-none-elf-
    stderrfilt: null
    params:
      PLAT: fvp
      BL33: ${artifact:EDK2}
      ARM_ARCH_MAJOR: 9
      CTX_INCLUDE_AARCH32_REGS: 0
      ARM_ARCH_MINOR: 2
      BRANCH_PROTECTION: 1
      FVP_HW_CONFIG_DTS: fdts/fvp-base-gicv3-psci-1t.dts
      LOG_LEVEL: 40
      ARM_DISABLE_TRUSTED_WDOG: 1
      DEBUG: 0
    prebuild: []
    build:
    - 'make BUILD_BASE=${param:builddir} ${param:join_equal} -j$$(( ${param:jobs}
      < 8 ? ${param:jobs} : 8 )) all fip'
    postbuild: []
    artifacts:
      FIP: ${param:builddir}/fvp/release/fip.bin
      BL1: ${param:builddir}/fvp/release/bl1.bin
      BL31: ${param:builddir}/fvp/release/bl31.bin
      BL2: ${param:builddir}/fvp/release/bl2.bin
buildex:
  btvars: {}
artifacts: {}
run:
  name: FVP_Base_RevC-2xAEMvA
  rtvars:
    CMDLINE:
      type: string
      value: console=ttyAMA0 earlycon=pl011,0x1c090000 root=/dev/vda ip=dhcp
    BL1:
      type: path
      value: ${artifact:BL1}
    DTB:
      type: path
      value: ${artifact:DTB}
    KERNEL:
      type: path
      value: null
    FIP:
      type: path
      value: ${artifact:FIP}
    EDK2FLASH:
```

(continues on next page)

(continued from previous page)

```
      type: path
      value: ''
    ROOTFS:
      type: path
      value: ''
    SHARE:
      type: path
      value: ''
    LOCAL_NET_PORT:
      type: string
      value: 8022
  params:
    -C cluster1.stage12_tlb_size: 1024
    -C cluster1.check_memory_attributes: 0
    -C cluster0.gicv4.mask-virtual-interrupt: 1
    -C bp.hostbridge.userNetworking: 1
    -C bp.flashloader0.fname: ${rtvar:FIP}
    -C pci.pci_smmuv3.mmu.SMMU_IDR1: 216481056
    -C cluster0.gicv3.without-DS-support: 1
    -C cluster1.has_arm_v8-3: 1
    -C cluster0.has_sve: 1
    -C cluster1.has_arm_v8-4: 1
    -C cluster0.sve.has_sme: 1
    -C cluster0.gicv3.cpuintf-mmap-access-level: 2
    -C cluster0.has_amu: 1
    -C cluster1.has_arm_v8-8: 1
    -C cluster1.has_brbe: 1
    -C bp.dram_size: 4
    -C cluster1.pmb_idr_external_abort: 1
    -C cluster1.has_arm_v9-0: 1
    -C bp.virtioblockdevice.image_path: ${rtvar:ROOTFS}
    -C cluster1.has_arm_v8-5: 1
    -C cluster0.has_arm_v8-3: 1
    -C cluster1.has_arm_v8-6: 1
    -C cluster1.has_hpmn0: 1
    -C bp.virtiop9device.root_path: ${rtvar:SHARE}
    -C cluster0.has_hpmn0: 1
    -C pci.pci_smmuv3.mmu.SMMU_IDR0: 135263935
    -C cluster1.NUM_CORES: 4
    -C cluster1.gicv3.cpuintf-mmap-access-level: 2
    -C cluster0.has_arm_v8-6: 1
    -C cluster1.has_amu: 1
    -C cluster1.enhanced_pac2_level: 3
    -C cache_state_modelled: 0
    -C cluster0.sve.has_sve2: 1
    -C cluster0.cpu0.semihosting-cwd: $${SEMIHOSTDIR}
    -C bp.virtio_rng.enabled: 1
    -C cluster0.has_arm_v9-0: 1
    -C cluster0.has_arm_v9-3: 1
    -C cluster1.sve.has_sve2: 1
    -C cluster0.has_rndr: 1
    -C cluster1.has_arm_v9-2: 1
```

(continues on next page)

(continued from previous page)

```
    -C gic_distributor.has_nmi: 1
    -C pci.pci_smmuv3.mmu.SMMU_S_IDR3: 0
    -C cluster0.has_brbe_v1p1: 1
    -C cluster0.has_branch_target_exception: 1
    -C cluster0.PA_SIZE: 48
    -C pci.pci_smmuv3.mmu.SMMU_S_IDR1: 2684354562
    -C cluster1.max_32bit_el: 0
    -C cluster0.has_arm_v9-2: 1
    -C cluster0.has_arm_v8-8: 1
    -C cluster1.PA_SIZE: 48
    -C cluster1.gicv3.without-DS-support: 1
    -C cluster0.has_arm_v9-1: 1
    -C cluster1.has_large_system_ext: 1
    -C cluster0.sve.has_sme2: 1
    -C cluster0.NUM_CORES: 4
    -C bp.secure_memory: 1
    -C bp.hostbridge.userNetPorts: ${rtvar:LOCAL_NET_PORT}=22
    -C cluster0.ecv_support_level: 2
    -C bp.flashloader1.fname: ${rtvar:EDK2FLASH}
    -C cluster1.sve.has_sme: 1
    -C cluster0.has_16k_granule: 1
    -C cluster0.has_large_system_ext: 1
    -C cluster1.has_rndr: 1
    -C cluster1.gicv4.mask-virtual-interrupt: 1
    -C bp.refcounter.non_arch_start_at_default: 1
    -C cluster0.pmb_idr_external_abort: 1
    -C pci.pci_smmuv3.mmu.SMMU_IDR5: 4294902901
    -C pci.pci_smmuv3.mmu.SMMU_S_IDR2: 0
    -C cluster1.has_arm_v8-1: 1
    -C cluster1.has_arm_v8-2: 1
    -C cluster1.has_branch_target_exception: 1
    --stat: null
    -C cluster0.has_arm_v8-4: 1
    -C cluster0.max_32bit_el: 0
    -C cluster1.has_16k_granule: 1
    -C cluster1.has_brbe_v1p1: 1
    -C cluster0.has_arm_v8-2: 1
    -C cluster1.has_const_pac: 1
    -C cluster0.enhanced_pac2_level: 3
    -C cluster1.clear_reg_top_eret: 2
    -C bp.vis.disable_visualisation: 1
    -C cluster0.has_arm_v8-1: 1
    -C bp.ve_sysregs.exit_on_shutdown: 1
    -C pctl.startup: 0.0.0.0
    -C cluster1.has_arm_v8-7: 1
    -C bp.smsc_91c111.enabled: 1
    -C cluster1.ecv_support_level: 2
    -C bp.secureflashloader.fname: ${rtvar:BL1}
    -C cluster0.has_arm_v8-5: 1
    -C cluster1.has_arm_v9-1: 1
    -C cluster1.has_sve: 1
    -C pci.pci_smmuv3.mmu.SMMU_AIDR: 2
```

(continues on next page)

(continued from previous page)

```
    -C cluster1.has_large_va: 1
    -C cluster0.has_const_pac: 1
    -C cluster0.clear_reg_top_eret: 2
    -C pci.pci_smmuv3.mmu.SMMU_IDR3: 5908
    -C cluster0.check_memory_attributes: 0
    -C cluster1.has_arm_v9-3: 1
    -C cluster1.sve.has_sme2: 1
    -C cluster0.stage12_tlb_size: 1024
    -C cluster0.has_arm_v8-7: 1
    -C cluster0.has_brbe: 1
    -C cluster0.has_large_va: 1
  prerun:
  - SEMIHOSTDIR=`mktemp -d`
  - function finish { rm -rf $$SEMIHOSTDIR; }
  - trap finish EXIT
  - cp ${rtvar:KERNEL} $${SEMIHOSTDIR}/Image
  - cp ${rtvar:DTB} $${SEMIHOSTDIR}/fdt.dtb
  - cat <<EOF > $${SEMIHOSTDIR}/startup.nsh
  - Image dtb=fdt.dtb ${rtvar:CMDLINE}
  - EOF
  run: []
  terminals:
    bp.terminal_1:
      port_regex: 'terminal_1: Listening for serial connection on port (\d+)'
      friendly: edk2
      type: stdout
    bp.terminal_2:
      friendly: term2
      port_regex: 'terminal_2: Listening for serial connection on port (\d+)'
      type: stdout
    bp.terminal_3:
      friendly: term3
      port_regex: 'terminal_3: Listening for serial connection on port (\d+)'
      type: stdout
    bp.terminal_0:
      friendly: ''
      type: stdinout
      no_escapes: 'EFI stub: Booting Linux Kernel...'
      port_regex: 'terminal_0: Listening for serial connection on port (\d+)'
      no_color: true
```

1.2.2 Run-Times

Shrinkwrap uses a "runtime" to execute all of its shell commands and allows the user to choose which runtime to use. Both the design and implementation of this is borrowed from Tuxmake.

Shrinkwrap supports the following set of runtimes:

run-time	description
null	Shell commands are executed natively on the user's system. The user is responsible for ensuring the the required toolchain, environment variables and any other dependencies are set up.
docker	(default). Shell commands are executed in a docker container. By default, the official shrinkwrap image will be pulled and used, which contains all dependencies already setup.
docker-local	Like docker, but will only look for the container image on the local system. Will not attempt to pull over the network.
podman	Shell commands are executed in a podman container. By default, the official shrinkwrap image will be pulled and used, which contains all dependencies already setup.
podman-local	Like podman, but will only look for the container image on the local system. Will not attempt to pull over the network.

The desired runtime can be specified using the `--runtime` option, which is a top-level argument (must come before the command):

```
shrinkwrap --runtime=<name> ...
```

If using a container runtime (anything other than null), a custom image can optionally be specified. If omitted, the official shrinkwrap image is used:

```
shrinkwrap --runtime=<name> --image=<name> ...
```

Container Image Variants

Shrinkwrap runs on both x86_64 and aarch64 architectures, and provides multiarch container images so that the correct variant is automatically selected for your platform. Images are automatically downloaded by shrinkwrap when the `docker` or `podman` runtime is selected. Images are available on Docker Hub and can be freely downloaded without the need for an account.

image name	description
docker.io/shrinkwraptool/base-slim-nofvp	Contains all toolchains and other dependencies required to build all standard configs. Can be used as a base to create an image with a custom FVP.
docker.io/shrinkwraptool/base-slim (default)	As per `shrinkwraptool/base-slim-nofvp` but also contains the Base_RevC-2xAEMvA FVP. This is sufficient for most use cases and is much smaller than the `full` variant.
docker.io/shrinkwraptool/base-full-nofvp	Builds upon `shrinkwraptool/base-slim`, adding aarch32 toolchains (both arm-none-eabi and arm-linux-gnueabihf). These are not needed for standard configs, but will be required if creating a custom config that includes (e.g.) SCP FW. Separated out due to big size increase.
docker.io/shrinkwraptool/base-full	As per `shrinkwraptool/base-full-nofvp` but also contains the Base_RevC-2xAEMvA FVP.

Container Image Tags

All container image variants are maintained with a tag corresponding to each shrinkwrap version, so tagged shrinkwrap releases always have a corresponding set of fixed, tagged container images (e.g. docker.io/shrinkwraptool/base-slim:2025.10.0). Additionally development versions are maintained (e.g. docker.io/shrinkwraptool/base-slim:2025.12.0.dev0), which are rebuilt as the development version matures until it reaches release, so the content of the development versions is not fixed. Finally, a `latest` tag is maintained, which corresponds to the most recent build of the current development version.

When invoking shrinkwrap, if an image is specified (either on command line via `--image` or in a config via `image:`) with a tag, it is used as is. If an image is specified without a tag, shrinkwrap will use it's current version as the tag.

Runtime Requirements

The best way to understand the requirements for the packages available within the runtime is to look at the dockerfiles for the official shrinkwrap images. These are available at `docker/Dockerfile.*`.

Build Container Image Locally

If you have a need to build the shrinkwrap container images on your local system, you can do it as follows:

```
cd docker
./build.sh --version local
```

This will build a set of images called:

- `shrinkwraptool/base-slim:local-<ARCH>`
- `shrinkwraptool/base-slim-nofvp:local-<ARCH>`
- `shrinkwraptool/base-full:local-<ARCH>`
- `shrinkwraptool/base-full-nofvp:local-<ARCH>`

To use a locally built image, call shrinkwrap as follows if running on an x86_64 system:

```
shrinkwrap --runtime=<name>-local --image=shrinkwraptool/base-slim:local-x86_64 ...
```

Or like this if running on an aarch64 system:

```
shrinkwrap --runtime=<name>-local --image=shrinkwraptool/base-slim:local-aarch64 ...
```

where <name> is either docker or podman. Note that because the image is not on Docker Hub, the <name>-local runtime is required to prevent Shrinkwrap from erroneously trying to download an update.

1.2.3 Commands

For documentation on the commands that shrinkwrap supports, add `--help` to the command line.

For top-level help:

```
shrinkwrap --help
```

For help on a specific command:

```
shrinkwrap inspect --help
shrinkwrap build --help
shrinkwrap buildall --help
shrinkwrap clean --help
shrinkwrap run --help
shrinkwrap process --help
```

Todo: Automate importing the help pages into this documentation.

1.2.4 Config Model

A config is a yaml file that defines everything about a given configuration. This includes:

- meta data (e.g. its name, description, etc)
- the components that should be built
- how those components are built
- dependencies between components
- what artifacts are produced
- how to configure the fvp
- how to load and run the artifacts on the fvp

A config is declarative; the user declares how things relate and how things should be done, and the tool extracts the information to decide exactly what should be done and in what order in order to complete a task. All the data is contained within the config and drives the tool. This way, shrinkwrap is highly extensible. The user specifies the config(s) that should be used when invoking shrinkwrap.

Merging Configs

A config is laid out as a hierarchical data structure, using nested dictionaries. This suits it very well to being split into partial configs that are merged together into a single, final config. This allows maximal reuse of the config fragments and improves maintainability. Each config can optionally define a set of foundational `layers` which it then builds upon. Furthermore, the user can optionally specify a custom `overlay` config on the command line. Layers are merged in order according to the following rules:

For each leaf key in the union of the hierarchical dictionaries:

- If the upper value is null or not present, then the lower value is taken
- If both the upper and lower values are lists, then the final value is the lower list with the upper list appended to its end.
- In all other cases the upper value is taken

You can use the `process` command to merge configs and see the resulting output to get a better feel for how it works. See *Commands*.

Merging Example

Listing 1: lower config

```
people:
  Iris:
    age: 2
    likes:
      - Peppa Pig
      - Bananas
```

Listing 2: upper config

```
people:
  Iris:
    age: 3
    likes:
      - Peas
  James:
    age: 6
    likes:
      - FIFA
```

Listing 3: merged result

```
people:
  Iris:
    age: 3
    likes:
      - Peppa Pig
      - Bananas
      - Peas
  James:
    age: 6
    likes:
      - FIFA
```

Macros

Macros are placeholders that can be specified in various parts of a config yaml file, which are substituted ("resolved") with information that is only known at build-time or run-time. There are specific rules about which macros can be used in which parts of the config, and about the order in which they get substituted.

Macros take the following form:

```
${<type>:[<name>]}
```

where:

- `type` is a required namespace for the macro family
- `name` is an optional name for the macro within its namespace. For some macro types, there are a fixed set of names. For others, the names are defined by the config itself.

You can use the `process` command to resolve macros and see the resulting output to get a better feel for how they work. See *Commands*.

Defined Macros

macro	scope	description
`${param:sourcedir}`	`sourcedir.<component>.{params, prebuild, build, postbuild, artifacts}`	Directory in which the component's source code is located.
`${param:builddir}`	`builddir.<component>.{params, prebuild, build, postbuild, artifacts}`	Directory in which the component should be built, if the component's build system supports separation of source and build trees.
`${param:configdir}`	`configdir.<component>.{params, prebuild, build, postbuild, artifacts}`	Directory containing the config store. This MUST only be used for resolving files that already exist in the store.
`${param:jobs}`	`build.<component>.{params, prebuild, build, postbuild, artifacts}`	Maximum number of low level parallel jobs specified on the command line. To be passed to (e.g.) make as `-j${param:jobs}`.
`${btvar:<name>}`	`name.<component>.{sourcedir, builddir, repo, toolchain, params, prebuild, build, postbuild, artifacts}, run.rtvars`	Build-time variables. The variable names, along with default values are declared in buildex.btvars, and the user may override the value on the command line.
`${param:join:<name>}`	`join.<component>.{prebuild, build, postbuild}`	String containing all of the component's parameters (from its params dictionary), concatenated as `key=value` pairs.
`${param:joinspace:<name>}`	`joinspace.<component>.{prebuild, build, postbuild}`	String containing all of the component's parameters (from its params dictionary), concatenated as `key value` pairs.
`${artifact:<name>}`	`build.<component>.{params, prebuild, build, postbuild, artifacts}, build.btvars`	Build path of an artifact declared by another component. Usage of these macros determine the component build dependency graph.
`${param:packagedir}`	`package.<name>.{params, artifacts}`	Root directory containing all component artifacts.
`${artifact:<name>}`	`run.rtvars`	Package path of an artifact.
`${rtvar:<name>}`	`name.params`	Run-time variables. The variable names, along with default values are declared in run.rtvars, and the user may override the value on the command line.

Schema

Top-Level keys

The following is the set of top-level public keys that should be defined by a config. There are some additional private keys that the tool will add (and make visible as part of the `process` command), but these are subject to change and not documented.

key	type	description
description	string	A human-readable description of what the config contains and does. Displayed by the `inspect` command.
image	string	An optional field to require a specific runtime image to build / run this config. If not defined, the default image used by `shrinkwrap` will be used. This can be overriden with the `--image` command line option to `shrinkwrap`.
concrete	boolean	true if the config is intended to be directly built and run, or false if it is intended as a fragment to be included in other configs.
build	dictionary	Contains all the components to be built. The key is the component name and the value is a dictionary.
run	dictionary	Contains all the information about how to run the exported artifacts on the FVP.

build section

The build section, contains a dictionary of components that must be built. The keys are the component names and the values are themselves dictionaries, each containing the component meta data.

buildex section

When the schema was originally created, we made a mistake. The components should have been under `build: components:`, allowing room for new build data to be added under `build:` without being confused for components. In order to retrofit a solution without breaking compatibility, the buildex section is created.

key	type	description
btvars	dictionary	Build-Time variables. Keys are the variable names and values are a dictionary with keys 'type' (which must be one of 'path' and 'string'), 'value' (which takes the default value), and 'options' (which is the list of allowed values, and can include 'null' which makes the variable optional). Build-Time variables can be overridden by the user at the command line.

component section

key	type	description
repo	dictionary	Specifies information about the git repo(s) that must be cloned and checked out. By default, Shrinkwrap syncs the git repo to the specified revision when building. `--no-sync`, `--no-sync-all` or the `sync` parameter can be used to tell Shrinkwrap to build it in whatever state the user left it in. Not required if `sourcedir` is provided.
sourcedir	string	If specified, points to the path on disk where the source repo can be found. Useful for developer use cases where a local repo already exists.
builddir	string	If specified, the location where the component will be built. If not specified, shrinkwrap allocates its own location based on SHRINKWRAP_BUILD.
toolchain	string	Defines the toolchain to be used for compilation. Value is set as CROSS_COMPILE environment variable before invoking any prebuild/build/postbuild commands. When using the standard image with a container runtime, the options are: `aarch64-none-elf-`, `arm-none-eabi-`, `aarch64-linux-gnu-`, or `arm-linux-gnueabihf-`.
stderrfilt	bool	Optional, defaults to false. When true, and –verbose is not specified, filters stderr of the component's build task so that only lines containing 'error' and 'warning' are output. Everything else is suppressed. Useful for EDK2 which is extremely chatty.
params	dictionary	Optional set of key:value pairs. When building most components, they require a set of parameters to be passed. By setting them out as a dictionary, it is easy to override and add to them in higher layers. See `${param:join_*}` macros.
prebuild	list	List of shell commands to be executed during component build before the `build` list.
build	list	List of shell commands to be executed during component build.
postbuild	list	List of shell commands to be executed during component build after the `build` list.
artifacts	dictionary	Set of artifacts (files and/or directories) that the component exports. Key is artifact name. Value is a dictionary where the key `path` points to the built artifact, `export` is a boolean determining whether it will be copied to the package directory, and `rename` is an optional string to which the artifact will be renamed during export; for brevity, value can instead be the path itself, in which case `export` is implied. Other components can reference them with the `${artifact:<name>}` macros. Used to determine build dependencies. Note that an artifact can only be used by the `run` section if it was exported by the `build` section.
sync	enum-string	Specifies how shrinkwrap should synchronize the repository. See below for options.

Sync mode:

- **true**: synchronize the source directory. Do not overwrite user modifications or download updates.
- **false**: do not synchronize the source directory.
- **force**: synchronize the source directory. Overwrite any user modification and download branch updates.

repo section

key	type	description
remote	string	Address of the remote repository
revision	string	A git revision (branch name, tag, hash…)
project	string	Optional. Name of the project corresponding to this repository, used to retrieve a cached repository in SHRINKWRAP_PROJECT_CACHE. By default, the project name is the base name contained in the remote address.

run section

key	type	description
name	string	Name of the FVP binary, which must be in $PATH.
rtvars	dictionary	Run-Time variables. Keys are the variable names and values are a dictionary with keys 'type' (which must be one of 'path' and 'string'), 'value' (which takes the default value), and 'options' (which is the list of allowed values, and can include 'null' which makes the variable optional). Run-Time variables can be overridden by the user at the command line.
params	dictionary	Dictionary of parameters to be passed to the FVP. Similar to the component's params, laying these out in a dictionary makes it easy for higher layers to override and add parameters.
prerun	list	List of shell commands to be executed before the FVP is started.
terminals	dictionary	Describes the set of UART terminals available for the FVP. key is the terminal parameter name known to the FVP (e.g. `bp.terminal_0`) See below for format of the value.

terminal section

key	type	description
friendly	string	Label to display against the terminal when muxing to stdout. An empty string disables the prefix for the output.
port_regex	string	Regex to use to find the TCP port of the terminal when parsing the FVP stdout. Must have single capture group.
type	enum-string	Terminal type. See below for options.
no_color	boolean	Optional (defaults to false, only applies to ['stdout', 'stdinout'] types): If true, output from this terminal is not color-coded. If this terminal carries the interactive shell, it is advised to set this to true to prevent interfering with the shell's escape sequences. –no-color command line option causes this to behave as if set to true.
no_escape_strip	boolean/string	Optional (defaults to false, only applies to ['stdout', 'stdinout'] types): If true, strips any escape sequences from the output stream before forwarding to the terminal. If a string, behaves as if true until the string is found in the output, which sets it to false. Useful to expunge escape sequences from EDK2 during boot.
logfile	string	Optional (defaults to none, only applies to ['stdout', 'stdinout'] types): Specifies path to a log file where all output to the terminal will be duplicated.

Terminal types:

- **stdout**: Mux output to stdout. Do not supply any input.
- **stdinout**: Mux output to stdout. Forward stdin to its input. Max of 1 of these types allowed.
- **telnet**: Shrinkwrap will print out a telnet command to run in a separate terminal to get a unique interactive terminal.
- **xterm**: Shrinkwrap will automatically launch xterm to provide a unique interactive terminal. Only works when runtime=null.

1.2.5 Config Store

Shrinkwrap ships with the following standard set of configs for use out-of-the-box:

bootwrapper.yaml

Description

Best choice for: I have a linux-system.axf boot-wrapper and want to run it.

Build to wrap a provided kernel with boot-wrapper EL3 FW into linux-system.axf. Provide the kernel image via the KERNEL btvar, and optionally override the kernel command line by providing the CMDLINE btvar.

Then run the boot-wrapper (or pass a separately created one as the BOOTWRAPPER rtvar) in the FVP. A ROOTFS can be optionally provided. If present it is loaded into the virtio block device (/dev/vda).

Build-Time Variables

btvar	default	options
CMDLINE	console=ttyAMA0 earlycon=pl011,0x1c090000 root=/dev/vda ip=dhcp	<required>
DTB	${artifact:DTB}	<required>
KERNEL	<null>	<required>

Run-Time Variables

rtvar	default	options
BOOTWRAPPER	${artifact:BOOTWRAPPER}	<required>
LOCAL_NET_PORT	8022	<required>
ROOTFS	<empty>	<required>

Components

component	repository	revision
bootwrapper	https://git.kernel.org/pub/scm/linux/kernel/git/mark/boot-wrapper-aarch64.git	master
dt	https://git.kernel.org/pub/scm/linux/kernel/git/devicetree/devicetree-rebasing.git	v6.19-dts

buildroot.yaml

Description

Generates a very simple rootfs as an ext2/4 image. Higher layers can modify the buildroot config by adding commands to prebuild.

Build-Time Variables

btvar	default	options

Run-Time Variables

rtvar	default	options

Components

component	repository	revision
buildroot	https://github.com/buildroot/buildroot.git	2026.02

cca-3world.yaml

Description

Brings together a software stack to demonstrate Arm CCA running on FVP in a three-world configuration. Includes TF-A in root world, RMM in realm world, and EDK2 and Linux in Normal world on the host. Guests can be launched in-realm in a number of configurations using kvmtool. EDK2 can be optionally used as guest FW.

If the user provides an ext2/4 filesystem image via the GUEST_ROOTFS btvar, a guest disk image is created that includes a FAT16 partition containing the guest kernel (to be loaded by the guest EDK2 FW), and the provided filesystem as the rootfs. The user can provide their own filesystem image, or alternatively use a simple buildroot image created with buildroot.yaml:

```
$ shrinkwrap build cca-3world.yaml --overlay buildroot.yaml --btvar GUEST_ROOTFS='$
↪{artifact:BUILDROOT}'
```

The user can also control the guest kernel command line parameters used on the guest disk image via the GUEST_CMDLINE btvar.

Once built, the user must get some of the generated artifacts into the FVP environment. This can either be done by copying them to the host's rootfs or by sharing them into the FVP using 9p.

For the time being, there is an issue in the linux kernel's handling of 9p which does not share correctly the guest image to the guest EFI, preventing the guest to boot. Copying the artifacts into the host's rootfs is the way to go. Something like the following example should work. For simplicity, this example reuses the guest filesystem generated with buildroot as the host's rootfs, after resizing it so that there is room for the guest's rootfs:

```
$ cd ~/.shrinkwrap/package/cca-3world
$ TOOLS_PATH=~/.shrinkwrap/build/build/cca-3world/buildroot/host/sbin
$ $TOOLS_PATH/e2fsck -fp rootfs.ext2
$ $TOOLS_PATH/resize2fs rootfs.ext2 256M
$ sudo su
# mkdir mnt
# mount rootfs.ext2 mnt
# mkdir mnt/cca
# cp guest-disk.img KVMTOOL_EFI.fd lkvm mnt/cca/.
# umount mnt
# rm -rf mnt
# exit
```

Now you can boot the host, using the rootfs we just modified, either using DT:

```
$ shrinkwrap run cca-3world.yaml --rtvar ROOTFS=rootfs.ext2
```

Or alternatively, using ACPI:

```
$ shrinkwrap run cca-3world.yaml -r ROOTFS=rootfs.ext2 --rtvar CMDLINE="mem=1G earlycon␣
↪root=/dev/vda ip=dhcp acpi=force"
```

Finally, once the host has booted, log in as "root" (no password), and launch a realm using kvmtool from the /cca directory (that was created above):

```
# cd /cca
# ./lkvm run --realm --disable-sve --irqchip=gicv3-its --firmware KVMTOOL_EFI.fd -c 1 -m␣
↪512 --no-pvtime --force-pci --disk guest-disk.img --measurement-algo=sha256 --
↪restricted_mem
```

Be patient while this boots to the UEFI shell. Navigate to "Boot Manager", then "UEFI Shell" and wait for the startup.nsh script to execute, which will launch the kernel. Continue to be patient, and eventually you will land at a login prompt. Login as "root" (no password).

When the linux kernel 9p issue will be fixed, the shared directory approach can be used. Simply boot the host with the SHARE rtvar. This only works for DT-based environments though:

```
$ cd ~/.shrinkwrap/package/cca-3world
$ shrinkwrap run cca-3world.yaml --rtvar ROOTFS=rootfs.ext2 --rtvar SHARE=.
```

Then, once the host has booted, log in as "root" (no password) and mount the shared folder to "/cca" and change dir to it. The realm guest can then be launched as previously:

```
# mkdir /cca
# mount -t 9p -o trans=virtio,version=9p2000.L FM /cca
```

(continues on next page)

(continued from previous page)

```
# cd /cca
# ./lkvm run --realm --disable-sve --irqchip=gicv3-its --firmware KVMTOOL_EFI.fd -c 1 -m␣
↪512 --no-pvtime --force-pci --disk guest-disk.img --measurement-algo=sha256 --
↪restricted_mem
```

It is also possible to launch Linux without using EDK2 as the guest FW:

```
# ./lkvm run --realm --disable-sve --irqchip=gicv3-its -c 1 -m 512 --no-pvtime --force-
↪pci --console virtio --kernel Image --disk guest-disk.img -p "console=hvc0 root=/dev/
↪vda2" --measurement-algo=sha256 --restricted_mem
```

This config also builds kvm-unit-tests, which can be run in the realm instead of Linux:

```
# cd /cca/kvm-unit-tests/arm
# export PATH=/cca:$PATH
# ./run-realm-tests
```

Build-Time Variables

btvar	default	options
EDK2_BUILD	RELEASE	DEBUG, RELEASE
GUEST_CMDLINE	root=/dev/vda2 acpi=force ip=on	<required>
GUEST_ROOTFS	<empty>	<required>
RMM_BASE	0xFDC00000	<required>
RMM_BUILD	Release	Debug, Release
RMM_LOG_LEVEL	40	<required>
TFA_BUILD	release	debug, release
TFA_LOG_LEVEL	40	<required>

Run-Time Variables

rtvar	default	options
BL1	${artifact:BL1}	<required>
CMDLINE	console=ttyAMA0 earlycon=pl011,0x1c090000 root=/dev/vda ip=dhcp	<required>
DTB	${artifact:DTB}	<required>
EDK2FLASH	<empty>	<required>
FIP	${artifact:FIP}	<required>
KERNEL	${artifact:KERNEL}	<required>
LOCAL_NET_PORT	8022	<required>
ROOTFS	<empty>	<required>
SHARE	<empty>	<required>

Components

component	repository	revision
acpica	https://github.com/acpica/acpica.git	20251212
dt	https://git.kernel.org/pub/scm/linux/kernel/git/devicetree/devicetree-rebasing.git	v6.19-dts
edk2	https://git.gitlab.arm.com/linux-arm/edk2-cca.git	3223_arm_cca_v4
edk2-cca-guest	https://git.gitlab.arm.com/linux-arm/edk2-cca.git	3223_arm_cca_v4
edk2-platforms	https://git.gitlab.arm.com/linux-arm/edk2-platforms-cca.git	3223_arm_cca_v4
kvm-unit-tests	https://gitlab.arm.com/linux-arm/kvm-unit-tests-cca	cca/rmm-v1.0-rel0
kvmtool (dtc)	https://git.kernel.org/pub/scm/utils/dtc/dtc.git	v1.6.1
kvmtool (kvm-tool)	https://gitlab.arm.com/linux-arm/kvmtool-cca	cca/v6
linux	https://git.gitlab.arm.com/linux-arm/linux-cca.git	cca-host/v8
rmm	https://git.trustedfirmware.org/TF-RMM/tf-rmm.git	tf-rmm-v0.8.0
tfa	https://git.trustedfirmware.org/TF-A/trusted-firmware-a.git	v2.14.0

cca-4world.yaml

Description

Builds on cca-3world.yaml, and adds support for running Hafnium along with some secure partitions in Secure World. Build with:

```
$ shrinkwrap build cca-4world.yaml --overlay buildroot.yaml --btvar GUEST_ROOTFS='$
↪{artifact:BUILDROOT}'
```

Then run the model with:

```
$ cd ~/.shrinkwrap/package/cca-4world
$ shrinkwrap run cca-4world.yaml --rtvar ROOTFS=rootfs.ext2 --rtvar SHARE=.
```

Once the host has booted, log in as "root" (no password).

Secure partitions can be enumerated by:

```
# cat /sys/devices/arm-ffa-*/uuid
b4b5671e-4a90-4fe1-b81f-fb13dae1dacb
d1582309-f023-47b9-827c-4464f5578fc8
79b55c73-1d8c-44b9-8593-61e1770ad8d2
eaba83d8-baaf-4eaf-8144-f7fdcbe544a7
```

See cca-3worlds.yaml config *Description* if willing to launch a realm using kvmtool.

Build-Time Variables

btvar	default	options
EDK2_BUILD	RELEASE	DEBUG, RELEASE
GUEST_CMDLINE	root=/dev/vda2 acpi=force ip=on	<required>
GUEST_ROOTFS	<empty>	<required>
RMM_BASE	0xFDC00000	<required>
RMM_BUILD	Release	Debug, Release
RMM_LOG_LEVEL	40	<required>
TFA_BUILD	release	debug, release
TFA_LOG_LEVEL	40	<required>
TFTF_BUILD	release	debug, release
TFTF_LOG_LEVEL	40	<required>
TFTF_SUITE	standard	<required>

Run-Time Variables

rtvar	default	options
BL1	${artifact:BL1}	<required>
CMDLINE	console=ttyAMA0 earlycon=pl011,0x1c090000 root=/dev/vda ip=dhcp	<required>
DTB	${artifact:DTB}	<required>
EDK2FLASH	<empty>	<required>
FIP	${artifact:FIP}	<required>
KERNEL	${artifact:KERNEL}	<required>
LOCAL_NET_PORT	8022	<required>
ROOTFS	<empty>	<required>
SHARE	<empty>	<required>

Components

component	repository	revision
acpica	https://github.com/acpica/acpica.git	20251212
dt	https://git.kernel.org/pub/scm/linux/kernel/git/devicetree/devicetree-rebasing.git	v6.19-dts
edk2	https://git.gitlab.arm.com/linux-arm/edk2-cca.git	3223_arm_cca_v4
edk2-cca-guest	https://git.gitlab.arm.com/linux-arm/edk2-cca.git	3223_arm_cca_v4
edk2-platforms	https://git.gitlab.arm.com/linux-arm/edk2-platforms-cca.git	3223_arm_cca_v4
hafnium	https://git.trustedfirmware.org/hafnium/hafnium.git	v2.14.0
kvm-unit-tests	https://gitlab.arm.com/linux-arm/kvm-unit-tests-cca	cca/rmm-v1.0-rel0
kvmtool (dtc)	https://git.kernel.org/pub/scm/utils/dtc/dtc.git	v1.6.1
kvmtool (kvm-tool)	https://gitlab.arm.com/linux-arm/kvmtool-cca	cca/v6
linux	https://git.gitlab.arm.com/linux-arm/linux-cca.git	cca-host/v8
rmm	https://git.trustedfirmware.org/TF-RMM/tf-rmm.git	tf-rmm-v0.8.0
tfa	https://git.trustedfirmware.org/TF-A/trusted-firmware-a.git	v2.14.0
tftf	https://git.trustedfirmware.org/TF-A/tf-a-tests.git	v2.14.0

cca-edk2.yaml

Description

Brings together TF-A, TF-RMM and EDK2 to provide a 3 world environment running on FVP. In this config TF-A is in Root World, TF-RMM is in Realm EL2 and EDK2 and Linux form the non-secure EL2. Allows easy specification of the kernel image and command line, and rootfs at runtime (see rtvars). ACPI is provided by UEFI. DT is enabled by default. Use 'acpi=force' command line option to enable ACPI boot.

By default (if not overriding the rtvars) a sensible command line is used that will set up the console for logging and attempt to mount the rootfs image from the FVP's virtio block device. However the default rootfs image is empty, so the kernel will panic when attempting to mount; the user must supply a rootfs if it is required that the kernel completes its boot. No default kernel image is supplied and the config will refuse to run unless it is explicitly specified.

Note that by default, UEFI variables are build time configured directing EDK2 to boot to the shell. This will cause startup.nsh to be executed and will start the kernel boot. This way everything is automatic. By default, all EDK2 output is muxed to stdout. If you prefer booting UEFI to its UI, override the the build pcd parameter *PcdUefiShellDefaultBootEnable* using the overlay and override terminals 'bp.terminal_0'.type to 'telnet'.

```
$ shrinkwrap build cca-edk2.yaml
```

```
$ shrinkwrap run cca-edk2.yaml --rtvar KERNEL=path/to/Image --rtvar ROOTFS=path/to/
↪rootfs.img
```

When booting with device tree, a directory can optionally be shared from the host system into the Linux environment running in the FVP. To do so, set the SHARE rtvar to the desired directory, then mount the share inside the FVP with the following (or automate it in fstab):

```
# mkdir /share
# mount -t 9p -o trans=virtio,version=9p2000.L FM /share
```

Build-Time Variables

btvar	default	options
EDK2_BUILD	RELEASE	DEBUG, RELEASE
RMM_BASE	0xFDC00000	<required>
RMM_BUILD	Release	Debug, Release
RMM_LOG_LEVEL	40	<required>
TFA_BUILD	release	debug, release
TFA_LOG_LEVEL	40	<required>

Run-Time Variables

rtvar	default	options
BL1	${artifact:BL1}	<required>
CMDLINE	console=ttyAMA0 earlycon=pl011,0x1c090000 root=/dev/vda ip=dhcp	<required>
DTB	${artifact:DTB}	<required>
EDK2FLASH	<empty>	<required>
FIP	${artifact:FIP}	<required>
KERNEL	<null>	<required>
LOCAL_NET_PORT	8022	<required>
ROOTFS	<empty>	<required>
SHARE	<empty>	<required>

Components

component	repository	revision
acpica	https://github.com/acpica/acpica.git	20251212
dt	https://git.kernel.org/pub/scm/linux/kernel/git/devicetree/devicetree-rebasing.git	v6.19-dts
edk2	https://git.gitlab.arm.com/linux-arm/edk2-cca.git	3223_arm_cca_v4
edk2-platforms	https://git.gitlab.arm.com/linux-arm/edk2-platforms-cca.git	3223_arm_cca_v4
rmm	https://git.trustedfirmware.org/TF-RMM/tf-rmm.git	tf-rmm-v0.8.0
tfa	https://git.trustedfirmware.org/TF-A/trusted-firmware-a.git	v2.14.0

ffa-hafnium-optee.yaml

Description

Brings together a software stack to demonstrate Arm FF-A running on FVP. Includes TF-A in secure EL3 running SPMD(Secure Partition Manager Dispatcher), Hafnium as secure Hypervisor at secure EL2 running SPMC (Secure Partition Manager Core) and OPTEE as a secure partition/VM in secure EL1 and Linux in Normal world.

Build-Time Variables

btvar	default	options
EDK2_BUILD	RELEASE	DEBUG, RELEASE
TFA_BUILD	release	debug, release
TFA_LOG_LEVEL	40	<required>

Run-Time Variables

rtvar	default	options
BL1	${artifact:BL1}	<required>
CMDLINE	console=ttyAMA0 earlycon=pl011,0x1c090000 root=/dev/vda ip=dhcp	<required>
DTB	${artifact:DTB}	<required>
EDK2FLASH	<empty>	<required>
FIP	${artifact:FIP}	<required>
KERNEL	<null>	<required>
LOCAL_NET_PORT	8022	<required>
ROOTFS	<empty>	<required>
SHARE	<empty>	<required>

Components

component	repository	revision
acpica	https://github.com/acpica/acpica.git	20251212
dt	https://git.kernel.org/pub/scm/linux/kernel/git/devicetree/devicetree-rebasing.git	v6.19-dts
edk2	https://github.com/tianocore/edk2.git	edk2-stable202511
edk2-platforms	https://github.com/tianocore/edk2-platforms.git	0f527ebcef4e111586bf7a62b0f24a7f6c7dc2a4
hafnium	https://git.trustedfirmware.org/hafnium/hafnium.git	v2.14.0
optee	https://github.com/OP-TEE/optee_os.git	4.9.0
tfa	https://git.trustedfirmware.org/TF-A/trusted-firmware-a.git	v2.14.0

ffa-optee.yaml

Description

Brings together a software stack to demonstrate Arm FF-A running on FVP. Includes TF-A in secure EL3 running SPMD(Secure Partition Manager Dispatcher), with secure EL2 disabled and SPMC(Secure Partition Manager Core) inside OPTEE at secure EL1 and Linux in Normal world.

Build-Time Variables

btvar	default	options
EDK2_BUILD	RELEASE	DEBUG, RELEASE
TFA_BUILD	release	debug, release
TFA_LOG_LEVEL	40	<required>

Run-Time Variables

rtvar	default	options
BL1	${artifact:BL1}	<required>
CMDLINE	console=ttyAMA0 earlycon=pl011,0x1c090000 root=/dev/vda ip=dhcp	<required>
DTB	${artifact:DTB}	<required>
EDK2FLASH	<empty>	<required>
FIP	${artifact:FIP}	<required>
KERNEL	<null>	<required>
LOCAL_NET_PORT	8022	<required>
ROOTFS	<empty>	<required>
SHARE	<empty>	<required>

Components

component	repository	revision
acpica	https://github.com/acpica/acpica.git	20251212
dt	https://git.kernel.org/pub/scm/linux/kernel/git/devicetree/devicetree-rebasing.git	v6.19-dts
edk2	https://github.com/tianocore/edk2.git	edk2-stable202511
edk2-platforms	https://github.com/tianocore/edk2-platforms.git	0f527ebcef4e111586bf7a62b0f24a7f6c7dc2a4
optee	https://github.com/OP-TEE/optee_os.git	4.9.0
tfa	https://git.trustedfirmware.org/TF-A/trusted-firmware-a.git	v2.14.0

ffa-tftf.yaml

Description

Brings together a software stack to demonstrate Arm FF-A running on FVP. Includes TF-A in secure EL3, Hafnium in secure EL2 and some demo TF-A test secure partitions.

Build-Time Variables

btvar	default	options
EDK2_BUILD	RELEASE	DEBUG, RELEASE
TFA_BUILD	release	debug, release
TFA_LOG_LEVEL	40	<required>
TFTF_BUILD	release	debug, release
TFTF_LOG_LEVEL	40	<required>
TFTF_SUITE	standard	<required>

Run-Time Variables

rtvar	default	options
BL1	${artifact:BL1}	<required>
CMDLINE	console=ttyAMA0 earlycon=pl011,0x1c090000 root=/dev/vda ip=dhcp	<required>
DTB	${artifact:DTB}	<required>
EDK2FLASH	<empty>	<required>
FIP	${artifact:FIP}	<required>
KERNEL	<null>	<required>
LOCAL_NET_PORT	8022	<required>
ROOTFS	<empty>	<required>
SHARE	<empty>	<required>

Components

component	repository	revision
acpica	https://github.com/acpica/acpica.git	20251212
dt	https://git.kernel.org/pub/scm/linux/kernel/git/devicetree/devicetree-rebasing.git	v6.19-dts
edk2	https://github.com/tianocore/edk2.git	edk2-stable202511
edk2-platforms	https://github.com/tianocore/edk2-platforms.git	0f527ebcef4e111586bf7a62b0f24a7f6c7dc2a4
hafnium	https://git.trustedfirmware.org/hafnium/hafnium.git	v2.14.0
tfa	https://git.trustedfirmware.org/TF-A/trusted-firmware-a.git	v2.14.0
tftf	https://git.trustedfirmware.org/TF-A/tf-a-tests.git	v2.14.0

ns-edk2-optee.yaml

Description

Brings together a software stack to demonstrate OPTEE in secure EL1 with TF-A in secure EL3 but without FF-A and secure EL2(Hafnium). Secure partition dispatcher exists inside OPTEE.

Build-Time Variables

btvar	default	options
EDK2_BUILD	RELEASE	DEBUG, RELEASE
TFA_BUILD	release	debug, release
TFA_LOG_LEVEL	40	<required>

Run-Time Variables

rtvar	default	options
BL1	${artifact:BL1}	<required>
CMDLINE	console=ttyAMA0 earlycon=pl011,0x1c090000 root=/dev/vda ip=dhcp	<required>
DTB	${artifact:DTB}	<required>
EDK2FLASH	<empty>	<required>
FIP	${artifact:FIP}	<required>
KERNEL	<null>	<required>
LOCAL_NET_PORT	8022	<required>
ROOTFS	<empty>	<required>
SHARE	<empty>	<required>

Components

component	repository	revision
acpica	https://github.com/acpica/acpica.git	20251212
dt	https://git.kernel.org/pub/scm/linux/kernel/git/devicetree/devicetree-rebasing.git	v6.19-dts
edk2	https://github.com/tianocore/edk2.git	edk2-stable202511
edk2-platforms	https://github.com/tianocore/edk2-platforms.git	0f527ebcef4e111586bf7a62b0f24a7f6c7dc2a4
optee	https://github.com/OP-TEE/optee_os.git	4.9.0
tfa	https://git.trustedfirmware.org/TF-A/trusted-firmware-a.git	v2.14.0

ns-edk2.yaml

Description

Best choice for: I want to run Linux on FVP, booting with ACPI/DT, and have easy control over its command line.

Brings together TF-A and EDK2 to provide a simple non-secure world environment running on FVP. Allows easy specification of the kernel image and command line, and rootfs at runtime (see rtvars). ACPI is provided by UEFI.

An extra rtvar is added (DTB) which allows specification of a custom device tree. By default (if not overriding the rtvar), the upstream kernel device tree is used. DT is enabled by default. Use 'acpi=force' to enable ACPI boot.

By default (if not overriding the rtvars) a sensible command line is used that will set up the console for logging and attempt to mount the rootfs image from the FVP's virtio block device. However the default rootfs image is empty, so the kernel will panic when attempting to mount; the user must supply a rootfs if it is required that the kernel completes its boot. No default kernel image is supplied and the config will refuse to run unless it is explicitly specified.

Note that by default, UEFI variables are build time configured directing EDK2 to boot to the shell. This will cause startup.nsh to be executed and will start the kernel boot. This way everything is automatic. By default, all EDK2 output is muxed to stdout. If you prefer booting UEFI to its UI, override the the build pcd parameter *PcdUefiShellDefaultBootEnable* using the overlay and override terminals 'bp.terminal_0'.type to 'telnet'.

When booting with device tree, a directory can optionally be shared from the host system into the Linux environment running in the FVP. To do so, set the SHARE rtvar to the desired directory, then mount the share inside the FVP with the following (or automate it in fstab):

```
# mkdir /share
# mount -t 9p -o trans=virtio,version=9p2000.L FM /share
```

Build-Time Variables

btvar	default	options
EDK2_BUILD	RELEASE	DEBUG, RELEASE
TFA_BUILD	release	debug, release
TFA_LOG_LEVEL	40	<required>

Run-Time Variables

rtvar	default	options
BL1	${artifact:BL1}	<required>
CMDLINE	console=ttyAMA0 earlycon=pl011,0x1c090000 root=/dev/vda ip=dhcp	<required>
DTB	${artifact:DTB}	<required>
EDK2FLASH	<empty>	<required>
FIP	${artifact:FIP}	<required>
KERNEL	<null>	<required>
LOCAL_NET_PORT	8022	<required>
ROOTFS	<empty>	<required>
SHARE	<empty>	<required>

Components

component	repository	revision
acpica	https://github.com/acpica/acpica.git	20251212
dt	https://git.kernel.org/pub/scm/linux/kernel/git/devicetree/devicetree-rebasing.git	v6.19-dts
edk2	https://github.com/tianocore/edk2.git	edk2-stable202511
edk2-platforms	https://github.com/tianocore/edk2-platforms.git	0f527ebcef4e111586bf7a62b0f24a7f6c7dc2a4
tfa	https://git.trustedfirmware.org/TF-A/trusted-firmware-a.git	v2.14.0

ns-preload.yaml

Description

Best choice for: I just want to run Linux on FVP.

A simple, non-secure-only configuration where all components are preloaded into memory (TF-A's BL31, DTB and kernel). The system resets directly to BL31. Allows easy specification of a custom command line at build-time (via build.dt.params dictionary) and specification of the device tree, kernel image and rootfs at run-time (see rtvars).

By default (if not overriding the rtvars), the upstream kernel device tree is used along with a sensible command line that will set up the console for logging and attempt to mount the rootfs image from the FVP's virtio block device. However the default rootfs image is empty, so the kernel will panic when attempting to mount; the user must supply a rootfs if it is required that the kernel completes its boot. No default kernel image is supplied and the config will refuse to run unless it is explicitly specified. Note: If specifying a custom dtb at runtime, this will also override any command line specified at build time, since the command line is added to the chosen node of the default dtb.

A directory can optionally be shared from the host system into the Linux environment running in the FVP. To do so, set the SHARE rtvar to the desired directory, then mount the share inside the FVP with the following (or automate it in fstab):

```
# mkdir /share
# mount -t 9p -o trans=virtio,version=9p2000.L FM /share
```

Build-Time Variables

btvar	default	options
TFA_BUILD	release	debug, release
TFA_LOG_LEVEL	40	<required>

Run-Time Variables

rtvar	default	options
BL31	${artifact:BL31}	<required>
DTB	${artifact:DTB}	<required>
KERNEL	<null>	<required>
LOCAL_NET_PORT	8022	<required>
ROOTFS	<empty>	<required>
SHARE	<empty>	<required>

Components

component	repository	revision
dt	https://git.kernel.org/pub/scm/linux/kernel/git/devicetree/devicetree-rebasing.git	v6.19-dts
tfa	https://git.trustedfirmware.org/TF-A/trusted-firmware-a.git	v2.14.0

rfa.yaml

Description

Rusted-Firmware-A. This configuration runs RF-A with the default features along with its normal-world and secure-world tests.

Build-Time Variables

btvar	default	options
TFA_BUILD	release	debug, release
TFA_LOG_LEVEL	40	<required>

Run-Time Variables

rtvar	default	options
BL1	${artifact:BL1}	<required>
FIP	${artifact:FIP}	<required>
LOCAL_NET_PORT	8022	<required>

Components

component	repository	revision
rfa	https://git.trustedfirmware.org/RF-A/rusted-firmware-a	v0.2.0
tfa	https://git.trustedfirmware.org/TF-A/trusted-firmware-a.git	v2.14.0

1.2.6 Shrinkwrap Recipes

This page contains an unordered list of Shrinkwrap usage examples intended to demonstrate how Shrinkwrap can solve various problems.

Override Component Version and/or Location

You can change many, many configuration options by overlaying a config on top of an existing config. Here we modify the revision and remote repository of the TF-A component from its default (defined in tfa-base.yaml). You could also specify the revision as a SHA or branch.

Create a file called `my-overlay.yaml`:

```
build:
  tfa:
    repo:
      remote: https://github.com/ARM-software/arm-trusted-firmware.git
      revision: v2.9
```

Optionally, you can view the final, merged config as follows:

```
shrinkwrap process --action=merge --overlay=my-overlay.yaml ns-preload.yaml
```

Now do a build, passing in the overlay:

```
shrinkwrap build --overlay=my-overlay.yaml ns-preload.yaml
```

Finally, boot the config:

```
shrinkwrap run --rtvar=KERNEL=path/to/Image ns-preload.yaml
```

Reuse Existing Local Repo for Component

By default, shrinkwrap will sync the git repos for all required components to a private location (<SHRINKWRAP_BUILD>/source/<config_name>/<component_name>) the first time you build a given config. However, sometimes you want shrinkwrap to reuse a repo that already exists on your local system. In this case, Shrinkwrap will build this source into its own private build tree, leaving the source tree unmodified.

Warning: Components support building in a tree separate from the source to differing degrees. For example, TF-A will always build fiptool in the source tree, although it will build all the FW components in the correct build tree. So depending on the component you are sharing source for, you may see some build artifacts appear.

Create a file called `my-overlay.yaml`:

```
build:
  tfa:
    sourcedir: /path/to/my/tfa/git/repo
    sync: false
```

Now do a build, passing in the overlay:

```
shrinkwrap build --overlay=my-overlay.yaml ns-preload.yaml
```

Changing Between Arm Architecture Revisions

Shrinkwrap comes with a set of configs that can be overlaid onto the primary config in order to modify the targeted Arm architecture revision. These overlays provide all the required modifications for the TF-A build configuration and the FVP run configuration. Each architecture revision includes all mandatory features associated with that extension as well as a selection of sensible/common optional features.

`Armv8.0` - `Armv8.8` and `Armv9.0` - `Armv9.3` are currently supported. The yaml files are in the `arch` subdirectory of the config store. (You can see them by running the `inspect` command with the `--all` option).

The below will build the `ns-edk2` config for Armv8.8 and run it on the FVP configured for the same revision.

```
shrinkwrap build ns-edk2.yaml --overlay=arch/v8.8.yaml
shrinkwrap run ns-edk2.yaml --rtvar=KERNEL=path/to/Image
```

Warning: Some components (notably TF-A) fail to incrementally build when changing their make parameters. Therefore, if you want to change the architecture revision for a config that has already been built, you must first clean tfa. See *Workaround for TF-A not Noticing Modified Build Params*.

Explicitly Clean a Config or Component

If the `build` command is invoked multiple times, Shrinkwrap will always attempt to do an incremental build. This enables a developer to modify the source and easily rebuild and run the result. However, sometimes it is useful to explicitly clean a component (or all the components within a config) to force it to be rebuilt from scratch. Shrinkwrap includes a `clean` command for this.

Clean an entire config (all components in config):

```
shrinkwrap clean ns-edk2.yaml
```

Clean a specific set of components from a config (in this case, clean the tfa and dt components):

```
shrinkwrap clean ns-edk2.yaml --filter=tfa --filter=dt
```

Then rebuild the config and the cleaned components are rebuilt from scratch:

```
shrinkwrap build ns-edk2.yaml
```

Workaround for TF-A not Noticing Modified Build Params

TF-A is not good at noticing when its build parameters change. If you have already built TF-A, then attempt to do an incremental build with different parameters, you rarely get what you expect. This happens a lot when using the arch/vX.Y.yaml overlays, because different architecture revisions need to specify different TF-A build parameters.

Work around this by explicitly cleaning TF-A when changing architecture revisions:

```
shrinkwrap build ns-edk2.yaml --overlay=arch/v8.7.yaml
shrinkwrap clean ns-edk2.yaml --filter=tfa
shrinkwrap build ns-edk2.yaml --overlay=arch/v9.3.yaml
```

Use a Custom FVP Version

By default, the `run` command will use the FVP that is bundled with the latest published shrinkwrap docker image. Sometimes you might want to use a different version though. In this case, the simplest approach is to install the FVP on your system, ensuring that the required directories are in your PATH, and invoke `shrinkwrap run` with the `null` runtime.

Shrinkwrap expects the FVP binary (e.g. FVP_Base_RevC-2xAEMvA) to be on your path. The example below shows downloading and untaring the FVP and adding the required directory to the PATH.

```
wget -q -O FVP_Base_RevC-2xAEMvA_11.18_16_Linux64.tgz https://developer.arm.com/-/media/
↪Files/downloads/ecosystem-models/FVP_Base_RevC-2xAEMvA_11.18_16_Linux64.tgz
tar xf FVP_Base_RevC-2xAEMvA_11.18_16_Linux64.tgz
export PATH=$PWD/Base_RevC_AEMvA_pkg/models/Linux64_GCC-9.3:$PATH
shrinkwrap build ns-edk2.yaml
shrinkwrap --runtime=null run ns-edk2.yaml --rtvar=KERNEL=path/to/Image
```

Use an Alternative Device Tree

There are a couple of ways to use an alternative device tree:

All provided concrete configs that use a device tree, expose a DTB rtvar with a default value. Users can override this value to provide an externally compiled DTB at **run-time**.

Alternatively, the dt-base.yaml config fragment can be passed a parameter at **build-time** that tells it to compile an alternative DTS file. dt-base.yaml builds the device tree for the FVP_Base_RevC-2xAEMvA FVP by default and is used by all the standard concrete configs that require a device tree and is available for use in defining custom configs.

Add the following to a higher layer of the config:

```
build:
  dt:
    prebuild:
      - DTS=foundation-v8-gicv3-psci.dts
```

Note that dt-base.yaml only accepts names of dts files that already exist in the device tree repo.

Accessing the FVP over Network when using Docker/Podman

When using the docker or podman runtimes, the FVP runs inside a container. This has a different IP address to the host system. Shrinkwrap helpfully prints out the runtime environment's IP address when starting the FVP. This is the IP address you need to use to (e.g.) connect the debugger or to SSH into the hosted Linux system.

Boot Linux with ACPI

`ns-edk2.yaml` uses EDK2 to boot Linux, and defaults to using the Device Tree. You can change the behaviour to boot with ACPI by passing `acpi=force` on the command line:

```
shrinkwrap run ns-edk2.yaml --rtvar=KERNEL=path/to/Image --rtvar=CMDLINE=
↪"console=ttyAMA0 earlycon=pl011,0x1c090000 root=/dev/vda ip=dhcp acpi=force"
```

Pass Overlay Directly on Command Line

All of the previous examples that utilize overlays, put the overlay in a yaml file and pass the file name on the command line. Here is an example that runs the FVP as normal but saves all output from UART0 to uart0.log:

Create a file called `my-overlay.yaml`:

```
run:
  params:
    -C bp.pl011_uart0.out_file: uart0.log
```

Now run the FVP, passing in the overlay:

```
shrinkwrap run ns-edk2.yaml --rtvar=KERNEL=path/to/Image --overlay=my-overlay.yaml
```

However, it is also possible to pass an overlay, encoded as json, directly on the command line, without the need for a file. This is useful when the overlay is small. JSON allows the entire content to be encoded on a single line without having to care about whitespace, so is suited to this purpose.

This is equivalent:

```
shrinkwrap run ns-edk2.yaml --rtvar=KERNEL=path/to/Image --overlay='{"run":{"params":{"-
↪C bp.pl011_uart0.out_file":"uart0.log"}}}'
```

Provide SSH keys

The GIT subprocess used by Shrinkwrap to synchronise source code may require access to SSH keys. This is supported, via `ssh-agent`.

If `ssh-agent` is already running, it will be automatically detected (via the `SSH_AUTH_SOCK` environment variable) and used by Shrinkwrap.

Alternatively, the `--ssh-agent` option can be used to request Shrinkwrap to start an `ssh-agent` subprocess and add default keys.

```
shrinkwrap --ssh-agent build ...
```

In order to add only specified keys, the `--ssh-agent-key` can be used.

```
shrinkwrap \
  --ssh-agent-key ~/.ssh/my-first-key \
  --ssh-agent-key ~/.ssh/my-second-key \
  build ...
```

1.3 Developer Guide

1.3.1 Release Process

Shrinkwrap is released at least once every quarter as a tagged snapshot of a branch. If critical bugs are discovered in the release, subsequent bugfix-only releases may be made available.

This document describes the process and serves as a checklist for the maintainer to make a shrinkwrap release.

Release Contents

Shrinkwrap consists of 3 parts:

- Shrinkwrap Python Package
- Shrinkwrap Container Images
- Shrinkwrap Documentation

A release is a consistent snapshot of all 3 parts, with a defined set of features that has been through a quality assurance process. The Python package is tagged in the Gitlab repository, the contianer images are tagged in docker hub, and the documentation is tagged in readthedocs. In future, the Python package may additionally be uploaded to PyPI.

Versioning Scheme

Shrinkwrap uses a date-based versioning scheme that is compatible with PEP 440, which is required by PyPI:

`<year>.<month>.<patch>[.dev0]`

component	description
`<year>`	Version year as 4 digit decimal integer.
`<month>`	Version month as 1 or 2 digit decimal integer (no trailing zero).
`<patch>`	Patch level: 0 for the primary release, then increments for each bugfix release.
`.dev0`	Used for in-development versions. A release snapshot never has this suffix.

As an example, let's say we have just released version `2025.12.0`. The version in the main branch of the repository will be immediately bumped to `2026.3.0.dev0` to indicate that the branch now represents the in-development version that will be released in March 2026. When development is complete and we are ready for release (hopefully in March 2026), the version in the main branch is updated to `2026.3.0` and the release is tagged. Then the version is updated to `2026.6.0.dev0` and the cycle begins again.

Bug Fix Releases

If the `<year>.<month>.0` release is found to have a critical bug, the decision may be taken to make a bugfix release which will increment the `<patch>`. First a branch is created in Gitlab, from the `<year>.<month>.0` version tag. The version is incremented to `<year>.<month>.1.dev0`, then fixes are backported from main onto the branch. Finally the version is set to `<year>.<month>.1` and the release is done following the same process but using the release branch.

Release Checklist

- Maintainer determines that `main` is ready to start the release process
  - All planned features have landed
  - The `main` CI pipeline passes
  - Discretionary manual testing has been performed and no issues found
  - Documentation has been reviewed and is up-to-date
- Maintainer writes and commits the release notes to `main`
- Maintainer makes commit to remove `.dev0` from version
- Maintainer triggers "full" pipeline in Gitlab, running full set of tests
  - On failure, debug and fix then go back to start
  - On success, containers are automatically tagged with version number in docker hub
- Maintainer creates tag of the tested SHA in Gitlab
- Maintainer creates new version corresponding to tag in readthedocs
- Maintainer makes commit to set the next version for the newly started development cycle `<year>.<month>.0.dev0`
- Maintainer and Project Manager fill out Arm-specific release forms
- Project Manager makes release announcement

1.3.2 Compile Documentation Locally

To build the docs locally, the following packages need to be installed on the host:

```
sudo apt-get install python3-pip
pip3 install -U -r documentation/requirements.txt
```

To build and generate the documentation in html format, run:

```
sphinx-build -b html -a -W documentation public
```

To render and explore the documentation, simply open *public/index.html* in a web browser.

1.4 Releases

1.4.1 2025.12.0

Links

- Tagged documentation: https://shrinkwrap.docs.arm.com/en/2025.12.0/
- Tagged source code: https://gitlab.arm.com/tooling/shrinkwrap/-/tags/2025.12.0
- Tagged container images:
  - shrinkwraptool/base-slim:2025.12.0
  - shrinkwraptool/base-full:2025.12.0

Component Versions

Toolchains

The base-slim container image includes:

- aarch64-linux-gnu-gcc (Debian 14.2.0-19) 14.2.0
- aarch64-none-elf-gcc (Arm GNU Toolchain 14.3.Rel1 (Build arm-14.174)) 14.3.1 20250623

The base-full container image additionally includes:

- arm-linux-gnueabihf-gcc (Debian 14.2.0-19) 14.2.0
- arm-none-eabi-gcc (Arm GNU Toolchain 14.3.Rel1 (Build arm-14.174)) 14.3.1 20250623
- Debian clang version 20.1.8 (++20250809043815+87f0227cb601-1~exp1~20250809163919.3)

FVP

The base-slim and base-full container images both include:

- FVP_Base_RevC-2xAEMvA [11.30.27 (Nov 14 2025)]

Software Components

A full list of every SW component, it's source repository and it's revision, is provided for each config that Shrinkwrap ships with. See *Config Store* for a list of configs.

Git Shortlog

- Chris Reed (2):
  - build: stringify string btvars with non-string values
  - refactor: use isinstance() instead of type()
- Mathias Brossard (1):
  - Add configuration files for running RF-A tests
- Olivier Deprez (3):
  - build: update to TF-A/Hafnium v2.14.0/TF-RMM v0.8.0
  - docker: bump AEM FVP to version 11.30 build 27
  - fvp: enable BRBE again
- Ryan Roberts (32):
  - version: Switch to new versioning scheme: 2025.12.0.dev0
  - docs: Document the release process
  - docs: Include table for all repos used in config
  - docs: Update auto-generated docs with repo info
  - docs: Display the version number within the documentation
  - containers: tag containers with shrinkwrap version
  - ci: Add option to control smoke vs full testing
  - ci: Don't run pipeline for tag or merge request events
  - test: Improve FVP output logging
  - config: Look for correct shutdown message with bootwrapper
  - config: Workaround bootwrapper's lack of support for GCS
  - config: Extend bootwrapper.yaml to support building
  - ci: Ensure caches get correctly invalidated
  - run: Don't show update_submodules() bash function
  - test: Honour config-defined container image
  - config: Update buildroot to 2025.08.1

  - config: Update dt to v6.17-dts
  - config: Update edk2 to edk2-stable202508.01
  - config: Update kvmtool revision
  - config: Update linux to v6.17
  - config: Update optee to 4.6.0
  - docker: Update to Debian Trixie as the container base image
  - docker: Update FVP to 11.29_42
  - docker: Update bare-metal toolchains to 14.3.rel1
  - docs: Update auto-generated docs for latest repo info
  - docs: Update config component versions
  - docs: Fix ReadTheDocs branch to version conversion
  - test: Bypass attestation checks when built on arm64
  - test: Save test output into log file
  - test: Fix test names to include overlays
  - test: Always keep test log
  - docs: Add framework for release notes
- Yousuf A (1):
  - docker: add e2fsprogs for rootdisk maintenance

Test Report Summary

All automated tests have been run and passed on the CI. This demonstrates that all configs build and boot correctly on the FVP.

There are 266 tests in total. All pass on both arm64 and x86_64 hosts.

- PASS: build:ns-preload.yaml:arch/v8.0.yaml
- PASS: build:ns-edk2.yaml:arch/v8.0.yaml
- PASS: build:ns-edk2-optee.yaml:arch/v8.0.yaml
- PASS: build:ffa-optee.yaml:arch/v8.0.yaml
- PASS: build:bootwrapper.yaml:arch/v8.0.yaml
- PASS: run:ns-preload.yaml:arch/v8.0.yaml
- PASS: run:ns-edk2.yaml:arch/v8.0.yaml:dt
- PASS: run:ns-edk2.yaml:arch/v8.0.yaml:acpi
- PASS: run:ns-edk2-optee.yaml:arch/v8.0.yaml
- PASS: run:ffa-optee.yaml:arch/v8.0.yaml:dt
- PASS: run:ffa-optee.yaml:arch/v8.0.yaml:acpi
- PASS: run:bootwrapper.yaml:arch/v8.0.yaml
- PASS: build:ns-preload.yaml:arch/v8.1.yaml

- PASS: build:ns-edk2.yaml:arch/v8.1.yaml
- PASS: build:ns-edk2-optee.yaml:arch/v8.1.yaml
- PASS: build:ffa-optee.yaml:arch/v8.1.yaml
- PASS: build:bootwrapper.yaml:arch/v8.1.yaml
- PASS: run:ns-preload.yaml:arch/v8.1.yaml
- PASS: run:ns-edk2.yaml:arch/v8.1.yaml:dt
- PASS: run:ns-edk2.yaml:arch/v8.1.yaml:acpi
- PASS: run:ns-edk2-optee.yaml:arch/v8.1.yaml
- PASS: run:ffa-optee.yaml:arch/v8.1.yaml:dt
- PASS: run:ffa-optee.yaml:arch/v8.1.yaml:acpi
- PASS: run:bootwrapper.yaml:arch/v8.1.yaml
- PASS: build:ns-preload.yaml:arch/v8.2.yaml
- PASS: build:ns-edk2.yaml:arch/v8.2.yaml
- PASS: build:ns-edk2-optee.yaml:arch/v8.2.yaml
- PASS: build:ffa-optee.yaml:arch/v8.2.yaml
- PASS: build:bootwrapper.yaml:arch/v8.2.yaml
- PASS: run:ns-preload.yaml:arch/v8.2.yaml
- PASS: run:ns-edk2.yaml:arch/v8.2.yaml:dt
- PASS: run:ns-edk2.yaml:arch/v8.2.yaml:acpi
- PASS: run:ns-edk2-optee.yaml:arch/v8.2.yaml
- PASS: run:ffa-optee.yaml:arch/v8.2.yaml:dt
- PASS: run:ffa-optee.yaml:arch/v8.2.yaml:acpi
- PASS: run:bootwrapper.yaml:arch/v8.2.yaml
- PASS: build:ns-preload.yaml:arch/v8.3.yaml
- PASS: build:ns-edk2.yaml:arch/v8.3.yaml
- PASS: build:ns-edk2-optee.yaml:arch/v8.3.yaml
- PASS: build:ffa-optee.yaml:arch/v8.3.yaml
- PASS: build:bootwrapper.yaml:arch/v8.3.yaml
- PASS: run:ns-preload.yaml:arch/v8.3.yaml
- PASS: run:ns-edk2.yaml:arch/v8.3.yaml:dt
- PASS: run:ns-edk2.yaml:arch/v8.3.yaml:acpi
- PASS: run:ns-edk2-optee.yaml:arch/v8.3.yaml
- PASS: run:ffa-optee.yaml:arch/v8.3.yaml:dt
- PASS: run:ffa-optee.yaml:arch/v8.3.yaml:acpi
- PASS: run:bootwrapper.yaml:arch/v8.3.yaml
- PASS: build:ns-preload.yaml:arch/v8.4.yaml

- PASS: build:ns-edk2.yaml:arch/v8.4.yaml
- PASS: build:ns-edk2-optee.yaml:arch/v8.4.yaml
- PASS: build:ffa-optee.yaml:arch/v8.4.yaml
- PASS: build:bootwrapper.yaml:arch/v8.4.yaml
- PASS: run:ns-preload.yaml:arch/v8.4.yaml
- PASS: run:ns-edk2.yaml:arch/v8.4.yaml:dt
- PASS: run:ns-edk2.yaml:arch/v8.4.yaml:acpi
- PASS: run:ns-edk2-optee.yaml:arch/v8.4.yaml
- PASS: run:ffa-optee.yaml:arch/v8.4.yaml:dt
- PASS: run:ffa-optee.yaml:arch/v8.4.yaml:acpi
- PASS: run:bootwrapper.yaml:arch/v8.4.yaml
- PASS: build:ns-preload.yaml:arch/v8.5.yaml
- PASS: build:ns-edk2.yaml:arch/v8.5.yaml
- PASS: build:ns-edk2-optee.yaml:arch/v8.5.yaml
- PASS: build:ffa-optee.yaml:arch/v8.5.yaml
- PASS: build:bootwrapper.yaml:arch/v8.5.yaml
- PASS: run:ns-preload.yaml:arch/v8.5.yaml
- PASS: run:ns-edk2.yaml:arch/v8.5.yaml:dt
- PASS: run:ns-edk2.yaml:arch/v8.5.yaml:acpi
- PASS: run:ns-edk2-optee.yaml:arch/v8.5.yaml
- PASS: run:ffa-optee.yaml:arch/v8.5.yaml:dt
- PASS: run:ffa-optee.yaml:arch/v8.5.yaml:acpi
- PASS: run:bootwrapper.yaml:arch/v8.5.yaml
- PASS: build:ns-preload.yaml:arch/v8.6.yaml
- PASS: build:ns-edk2.yaml:arch/v8.6.yaml
- PASS: build:ns-edk2-optee.yaml:arch/v8.6.yaml
- PASS: build:ffa-optee.yaml:arch/v8.6.yaml
- PASS: build:bootwrapper.yaml:arch/v8.6.yaml
- PASS: run:ns-preload.yaml:arch/v8.6.yaml
- PASS: run:ns-edk2.yaml:arch/v8.6.yaml:dt
- PASS: run:ns-edk2.yaml:arch/v8.6.yaml:acpi
- PASS: run:ns-edk2-optee.yaml:arch/v8.6.yaml
- PASS: run:ffa-optee.yaml:arch/v8.6.yaml:dt
- PASS: run:ffa-optee.yaml:arch/v8.6.yaml:acpi
- PASS: run:bootwrapper.yaml:arch/v8.6.yaml
- PASS: build:ns-preload.yaml:arch/v8.7.yaml

- PASS: build:ns-edk2.yaml:arch/v8.7.yaml
- PASS: build:ns-edk2-optee.yaml:arch/v8.7.yaml
- PASS: build:ffa-optee.yaml:arch/v8.7.yaml
- PASS: build:bootwrapper.yaml:arch/v8.7.yaml
- PASS: run:ns-preload.yaml:arch/v8.7.yaml
- PASS: run:ns-edk2.yaml:arch/v8.7.yaml:dt
- PASS: run:ns-edk2.yaml:arch/v8.7.yaml:acpi
- PASS: run:ns-edk2-optee.yaml:arch/v8.7.yaml
- PASS: run:ffa-optee.yaml:arch/v8.7.yaml:dt
- PASS: run:ffa-optee.yaml:arch/v8.7.yaml:acpi
- PASS: run:bootwrapper.yaml:arch/v8.7.yaml
- PASS: build:ns-preload.yaml:arch/v8.8.yaml
- PASS: build:ns-edk2.yaml:arch/v8.8.yaml
- PASS: build:ns-edk2-optee.yaml:arch/v8.8.yaml
- PASS: build:ffa-optee.yaml:arch/v8.8.yaml
- PASS: build:bootwrapper.yaml:arch/v8.8.yaml
- PASS: run:ns-preload.yaml:arch/v8.8.yaml
- PASS: run:ns-edk2.yaml:arch/v8.8.yaml:dt
- PASS: run:ns-edk2.yaml:arch/v8.8.yaml:acpi
- PASS: run:ns-edk2-optee.yaml:arch/v8.8.yaml
- PASS: run:ffa-optee.yaml:arch/v8.8.yaml:dt
- PASS: run:ffa-optee.yaml:arch/v8.8.yaml:acpi
- PASS: run:bootwrapper.yaml:arch/v8.8.yaml
- PASS: build:ns-preload.yaml:arch/v8.9.yaml
- PASS: build:ns-edk2.yaml:arch/v8.9.yaml
- PASS: build:ns-edk2-optee.yaml:arch/v8.9.yaml
- PASS: build:ffa-optee.yaml:arch/v8.9.yaml
- PASS: build:bootwrapper.yaml:arch/v8.9.yaml
- PASS: run:ns-preload.yaml:arch/v8.9.yaml
- PASS: run:ns-edk2.yaml:arch/v8.9.yaml:dt
- PASS: run:ns-edk2.yaml:arch/v8.9.yaml:acpi
- PASS: run:ns-edk2-optee.yaml:arch/v8.9.yaml
- PASS: run:ffa-optee.yaml:arch/v8.9.yaml:dt
- PASS: run:ffa-optee.yaml:arch/v8.9.yaml:acpi
- PASS: run:bootwrapper.yaml:arch/v8.9.yaml
- PASS: build:ns-preload.yaml:arch/v9.0.yaml

- PASS: build:ns-edk2.yaml:arch/v9.0.yaml
- PASS: build:ns-edk2-optee.yaml:arch/v9.0.yaml
- PASS: build:ffa-optee.yaml:arch/v9.0.yaml
- PASS: build:bootwrapper.yaml:arch/v9.0.yaml
- PASS: run:ns-preload.yaml:arch/v9.0.yaml
- PASS: run:ns-edk2.yaml:arch/v9.0.yaml:dt
- PASS: run:ns-edk2.yaml:arch/v9.0.yaml:acpi
- PASS: run:ns-edk2-optee.yaml:arch/v9.0.yaml
- PASS: run:ffa-optee.yaml:arch/v9.0.yaml:dt
- PASS: run:ffa-optee.yaml:arch/v9.0.yaml:acpi
- PASS: run:bootwrapper.yaml:arch/v9.0.yaml
- PASS: build:ns-preload.yaml:arch/v9.1.yaml
- PASS: build:ns-edk2.yaml:arch/v9.1.yaml
- PASS: build:ns-edk2-optee.yaml:arch/v9.1.yaml
- PASS: build:ffa-optee.yaml:arch/v9.1.yaml
- PASS: build:bootwrapper.yaml:arch/v9.1.yaml
- PASS: run:ns-preload.yaml:arch/v9.1.yaml
- PASS: run:ns-edk2.yaml:arch/v9.1.yaml:dt
- PASS: run:ns-edk2.yaml:arch/v9.1.yaml:acpi
- PASS: run:ns-edk2-optee.yaml:arch/v9.1.yaml
- PASS: run:ffa-optee.yaml:arch/v9.1.yaml:dt
- PASS: run:ffa-optee.yaml:arch/v9.1.yaml:acpi
- PASS: run:bootwrapper.yaml:arch/v9.1.yaml
- PASS: build:ns-preload.yaml:arch/v9.2.yaml
- PASS: build:ns-edk2.yaml:arch/v9.2.yaml
- PASS: build:ns-edk2-optee.yaml:arch/v9.2.yaml
- PASS: build:ffa-optee.yaml:arch/v9.2.yaml
- PASS: build:bootwrapper.yaml:arch/v9.2.yaml
- PASS: run:ns-preload.yaml:arch/v9.2.yaml
- PASS: run:ns-edk2.yaml:arch/v9.2.yaml:dt
- PASS: run:ns-edk2.yaml:arch/v9.2.yaml:acpi
- PASS: run:ns-edk2-optee.yaml:arch/v9.2.yaml
- PASS: run:ffa-optee.yaml:arch/v9.2.yaml:dt
- PASS: run:ffa-optee.yaml:arch/v9.2.yaml:acpi
- PASS: run:bootwrapper.yaml:arch/v9.2.yaml
- PASS: build:ns-preload.yaml:arch/v9.3.yaml

- PASS: build:ns-edk2.yaml:arch/v9.3.yaml
- PASS: build:ns-edk2-optee.yaml:arch/v9.3.yaml
- PASS: build:ffa-optee.yaml:arch/v9.3.yaml
- PASS: build:bootwrapper.yaml:arch/v9.3.yaml
- PASS: run:ns-preload.yaml:arch/v9.3.yaml
- PASS: run:ns-edk2.yaml:arch/v9.3.yaml:dt
- PASS: run:ns-edk2.yaml:arch/v9.3.yaml:acpi
- PASS: run:ns-edk2-optee.yaml:arch/v9.3.yaml
- PASS: run:ffa-optee.yaml:arch/v9.3.yaml:dt
- PASS: run:ffa-optee.yaml:arch/v9.3.yaml:acpi
- PASS: run:bootwrapper.yaml:arch/v9.3.yaml
- PASS: build:ns-preload.yaml:arch/v9.4.yaml
- PASS: build:ns-edk2.yaml:arch/v9.4.yaml
- PASS: build:ns-edk2-optee.yaml:arch/v9.4.yaml
- PASS: build:ffa-optee.yaml:arch/v9.4.yaml
- PASS: run:ns-preload.yaml:arch/v9.4.yaml
- PASS: run:ns-edk2.yaml:arch/v9.4.yaml:dt
- PASS: run:ns-edk2.yaml:arch/v9.4.yaml:acpi
- PASS: run:ns-edk2-optee.yaml:arch/v9.4.yaml
- PASS: run:ffa-optee.yaml:arch/v9.4.yaml:dt
- PASS: run:ffa-optee.yaml:arch/v9.4.yaml:acpi
- PASS: build:ns-preload.yaml:arch/v9.5.yaml
- PASS: build:ns-edk2.yaml:arch/v9.5.yaml
- PASS: build:ns-edk2-optee.yaml:arch/v9.5.yaml
- PASS: build:ffa-optee.yaml:arch/v9.5.yaml
- PASS: run:ns-preload.yaml:arch/v9.5.yaml
- PASS: run:ns-edk2.yaml:arch/v9.5.yaml:dt
- PASS: run:ns-edk2.yaml:arch/v9.5.yaml:acpi
- PASS: run:ns-edk2-optee.yaml:arch/v9.5.yaml
- PASS: run:ffa-optee.yaml:arch/v9.5.yaml:dt
- PASS: run:ffa-optee.yaml:arch/v9.5.yaml:acpi
- PASS: build:ffa-hafnium-optee.yaml:arch/v8.5.yaml
- PASS: build:ffa-tftf.yaml:arch/v8.5.yaml
- PASS: run:ffa-hafnium-optee.yaml:arch/v8.5.yaml:dt
- PASS: run:ffa-hafnium-optee.yaml:arch/v8.5.yaml:acpi
- PASS: run:ffa-tftf.yaml:arch/v8.5.yaml:dt

- PASS: run:ffa-tftf.yaml:arch/v8.5.yaml:acpi
- PASS: build:ffa-hafnium-optee.yaml:arch/v8.6.yaml
- PASS: build:ffa-tftf.yaml:arch/v8.6.yaml
- PASS: run:ffa-hafnium-optee.yaml:arch/v8.6.yaml:dt
- PASS: run:ffa-hafnium-optee.yaml:arch/v8.6.yaml:acpi
- PASS: run:ffa-tftf.yaml:arch/v8.6.yaml:dt
- PASS: run:ffa-tftf.yaml:arch/v8.6.yaml:acpi
- PASS: build:ffa-hafnium-optee.yaml:arch/v8.7.yaml
- PASS: build:ffa-tftf.yaml:arch/v8.7.yaml
- PASS: run:ffa-hafnium-optee.yaml:arch/v8.7.yaml:dt
- PASS: run:ffa-hafnium-optee.yaml:arch/v8.7.yaml:acpi
- PASS: run:ffa-tftf.yaml:arch/v8.7.yaml:dt
- PASS: run:ffa-tftf.yaml:arch/v8.7.yaml:acpi
- PASS: build:ffa-hafnium-optee.yaml:arch/v8.8.yaml
- PASS: build:ffa-tftf.yaml:arch/v8.8.yaml
- PASS: run:ffa-hafnium-optee.yaml:arch/v8.8.yaml:dt
- PASS: run:ffa-hafnium-optee.yaml:arch/v8.8.yaml:acpi
- PASS: run:ffa-tftf.yaml:arch/v8.8.yaml:dt
- PASS: run:ffa-tftf.yaml:arch/v8.8.yaml:acpi
- PASS: build:ffa-hafnium-optee.yaml:arch/v8.9.yaml
- PASS: build:ffa-tftf.yaml:arch/v8.9.yaml
- PASS: run:ffa-hafnium-optee.yaml:arch/v8.9.yaml:dt
- PASS: run:ffa-hafnium-optee.yaml:arch/v8.9.yaml:acpi
- PASS: run:ffa-tftf.yaml:arch/v8.9.yaml:dt
- PASS: run:ffa-tftf.yaml:arch/v8.9.yaml:acpi
- PASS: build:ffa-hafnium-optee.yaml:arch/v9.0.yaml
- PASS: build:ffa-tftf.yaml:arch/v9.0.yaml
- PASS: run:ffa-hafnium-optee.yaml:arch/v9.0.yaml:dt
- PASS: run:ffa-hafnium-optee.yaml:arch/v9.0.yaml:acpi
- PASS: run:ffa-tftf.yaml:arch/v9.0.yaml:dt
- PASS: run:ffa-tftf.yaml:arch/v9.0.yaml:acpi
- PASS: build:ffa-hafnium-optee.yaml:arch/v9.1.yaml
- PASS: build:ffa-tftf.yaml:arch/v9.1.yaml
- PASS: run:ffa-hafnium-optee.yaml:arch/v9.1.yaml:dt
- PASS: run:ffa-hafnium-optee.yaml:arch/v9.1.yaml:acpi
- PASS: run:ffa-tftf.yaml:arch/v9.1.yaml:dt

- PASS: run:ffa-tftf.yaml:arch/v9.1.yaml:acpi
- PASS: build:ffa-hafnium-optee.yaml:arch/v9.2.yaml
- PASS: build:ffa-tftf.yaml:arch/v9.2.yaml
- PASS: run:ffa-hafnium-optee.yaml:arch/v9.2.yaml:dt
- PASS: run:ffa-hafnium-optee.yaml:arch/v9.2.yaml:acpi
- PASS: run:ffa-tftf.yaml:arch/v9.2.yaml:dt
- PASS: run:ffa-tftf.yaml:arch/v9.2.yaml:acpi
- PASS: build:ffa-hafnium-optee.yaml:arch/v9.3.yaml
- PASS: build:ffa-tftf.yaml:arch/v9.3.yaml
- PASS: run:ffa-hafnium-optee.yaml:arch/v9.3.yaml:dt
- PASS: run:ffa-hafnium-optee.yaml:arch/v9.3.yaml:acpi
- PASS: run:ffa-tftf.yaml:arch/v9.3.yaml:dt
- PASS: run:ffa-tftf.yaml:arch/v9.3.yaml:acpi
- PASS: build:ffa-hafnium-optee.yaml:arch/v9.4.yaml
- PASS: build:ffa-tftf.yaml:arch/v9.4.yaml
- PASS: run:ffa-hafnium-optee.yaml:arch/v9.4.yaml:dt
- PASS: run:ffa-hafnium-optee.yaml:arch/v9.4.yaml:acpi
- PASS: run:ffa-tftf.yaml:arch/v9.4.yaml:dt
- PASS: run:ffa-tftf.yaml:arch/v9.4.yaml:acpi
- PASS: build:ffa-hafnium-optee.yaml:arch/v9.5.yaml
- PASS: build:ffa-tftf.yaml:arch/v9.5.yaml
- PASS: run:ffa-hafnium-optee.yaml:arch/v9.5.yaml:dt
- PASS: run:ffa-hafnium-optee.yaml:arch/v9.5.yaml:acpi
- PASS: run:ffa-tftf.yaml:arch/v9.5.yaml:dt
- PASS: run:ffa-tftf.yaml:arch/v9.5.yaml:acpi
- PASS: build:cca-3world.yaml
- PASS: build:cca-edk2.yaml
- PASS: build:rfa.yaml
- PASS: run:cca-3world.yaml
- PASS: run:cca-edk2.yaml:dt
- PASS: run:cca-edk2.yaml:acpi
- PASS: run:rfa.yaml
- PASS: build:cca-3world.yaml:buildroot-cca.yaml,test/cca.yaml
- PASS: run:cca-3world.yaml:buildroot-cca.yaml,test/cca.yaml:realm
- PASS: build:cca-4world.yaml
- PASS: run:cca-4world.yaml

- PASS: repo-sync-behaviours

1.4.2 2026.3.0

Links

- Tagged documentation: https://shrinkwrap.docs.arm.com/en/2026.3.0/
- Tagged source code: https://gitlab.arm.com/tooling/shrinkwrap/-/tags/2026.3.0
- Tagged container images:
  - shrinkwraptool/base-slim:2026.3.0
  - shrinkwraptool/base-full:2026.3.0

Component Versions

Toolchains

The base-slim container image includes:

- aarch64-linux-gnu-gcc (Debian 14.2.0-19) 14.2.0
- aarch64-none-elf-gcc (Arm GNU Toolchain 15.2.Rel1 (Build arm-15.86)) 15.2.1 20251203

The base-full container image additionally includes:

- arm-linux-gnueabihf-gcc (Debian 14.2.0-19) 14.2.0
- arm-none-eabi-gcc (Arm GNU Toolchain 15.2.Rel1 (Build arm-15.86)) 15.2.1 20251203
- Debian clang version 20.1.8 (++20250809043815+87f0227cb601-1~exp1~20250809163919.3)

FVP

The base-slim and base-full container images both include:

- FVP_Base_RevC-2xAEMvA [11.31.28 (Mar 1 2026)]

Software Components

A full list of every SW component, it's source repository and it's revision, is provided for each config that Shrinkwrap ships with. See *Config Store* for a list of configs. (Ensure you are viewing the version of that page that corresponds to this release).

Git Shortlog

- Chris Reed (1):
  - fix: handle empty ssh_agent_keys in Runtime.__init__()
- Mark Brown (2):
  - config: Add v9.6 architecture support
  - config: Add v9.7 architecture support
- Mathias Brossard (1):
  - ci: work-around for LLVM package issue
- Olivier Deprez (1):
  - RF-A bump to release v0.2.0
- Ryan Roberts (13):
  - version: Start 2026.3.0 development cycle
  - docker: Ensure we always have up-to-date package info
  - ci: Serialize interaction with ReadTheDocs
  - ci: Delete .cache_exists when invalidating test asset cache
  - docker: Update FVP to 11.31_28
  - docker: Update bare-metal toolchains to 15.2.rel1
  - config: Update buildroot to 2026.02
  - config: Update dt to v6.19-dts
  - config: Update edk2 to edk2-stable202511
  - config: Update kvmtool revision
  - config: Update linux to v6.19
  - config: Update optee to 4.9.0
  - docs: Update config component versions
- Saul Romero D (2):
  - fix: Bugfix for inspect command
  - fix: Remove default type for overlay variables
- Varshit Pandya (1):
  - fix: Replace GCC5 with GCC
- Yuliang Wang (9):
  - config: Optional list of allowed values for btvars and rtvars
  - config: Use btvars to switch between Debug / Release builds
  - config: Auto-generate debugger script
  - docs: Support for btvar/rtvar options and update config info
  - config: fix Trusted Firmware clean builds
  - config: Split edk2 and edk2-platforms into independent targets

  - config: Make edk2-base re-usable
  - docs: Updates related to EDK-II changes
  - fix(config): CCA EDK-II guest remote

Test Report Summary

All automated tests have been run and passed on the CI. This demonstrates that all configs build and boot correctly on the FVP.

There are 266 tests in total. All pass on both arm64 and x86_64 hosts.

- PASS: build:ns-preload.yaml:arch/v8.0.yaml
- PASS: build:ns-edk2.yaml:arch/v8.0.yaml
- PASS: build:ns-edk2-optee.yaml:arch/v8.0.yaml
- PASS: build:ffa-optee.yaml:arch/v8.0.yaml
- PASS: build:bootwrapper.yaml:arch/v8.0.yaml
- PASS: run:ns-preload.yaml:arch/v8.0.yaml
- PASS: run:ns-edk2.yaml:arch/v8.0.yaml:dt
- PASS: run:ns-edk2.yaml:arch/v8.0.yaml:acpi
- PASS: run:ns-edk2-optee.yaml:arch/v8.0.yaml
- PASS: run:ffa-optee.yaml:arch/v8.0.yaml:dt
- PASS: run:ffa-optee.yaml:arch/v8.0.yaml:acpi
- PASS: run:bootwrapper.yaml:arch/v8.0.yaml
- PASS: build:ns-preload.yaml:arch/v8.1.yaml
- PASS: build:ns-edk2.yaml:arch/v8.1.yaml
- PASS: build:ns-edk2-optee.yaml:arch/v8.1.yaml
- PASS: build:ffa-optee.yaml:arch/v8.1.yaml
- PASS: build:bootwrapper.yaml:arch/v8.1.yaml
- PASS: run:ns-preload.yaml:arch/v8.1.yaml
- PASS: run:ns-edk2.yaml:arch/v8.1.yaml:dt
- PASS: run:ns-edk2.yaml:arch/v8.1.yaml:acpi
- PASS: run:ns-edk2-optee.yaml:arch/v8.1.yaml
- PASS: run:ffa-optee.yaml:arch/v8.1.yaml:dt
- PASS: run:ffa-optee.yaml:arch/v8.1.yaml:acpi
- PASS: run:bootwrapper.yaml:arch/v8.1.yaml
- PASS: build:ns-preload.yaml:arch/v8.2.yaml
- PASS: build:ns-edk2.yaml:arch/v8.2.yaml
- PASS: build:ns-edk2-optee.yaml:arch/v8.2.yaml
- PASS: build:ffa-optee.yaml:arch/v8.2.yaml

- PASS: build:bootwrapper.yaml:arch/v8.2.yaml
- PASS: run:ns-preload.yaml:arch/v8.2.yaml
- PASS: run:ns-edk2.yaml:arch/v8.2.yaml:dt
- PASS: run:ns-edk2.yaml:arch/v8.2.yaml:acpi
- PASS: run:ns-edk2-optee.yaml:arch/v8.2.yaml
- PASS: run:ffa-optee.yaml:arch/v8.2.yaml:dt
- PASS: run:ffa-optee.yaml:arch/v8.2.yaml:acpi
- PASS: run:bootwrapper.yaml:arch/v8.2.yaml
- PASS: build:ns-preload.yaml:arch/v8.3.yaml
- PASS: build:ns-edk2.yaml:arch/v8.3.yaml
- PASS: build:ns-edk2-optee.yaml:arch/v8.3.yaml
- PASS: build:ffa-optee.yaml:arch/v8.3.yaml
- PASS: build:bootwrapper.yaml:arch/v8.3.yaml
- PASS: run:ns-preload.yaml:arch/v8.3.yaml
- PASS: run:ns-edk2.yaml:arch/v8.3.yaml:dt
- PASS: run:ns-edk2.yaml:arch/v8.3.yaml:acpi
- PASS: run:ns-edk2-optee.yaml:arch/v8.3.yaml
- PASS: run:ffa-optee.yaml:arch/v8.3.yaml:dt
- PASS: run:ffa-optee.yaml:arch/v8.3.yaml:acpi
- PASS: run:bootwrapper.yaml:arch/v8.3.yaml
- PASS: build:ns-preload.yaml:arch/v8.4.yaml
- PASS: build:ns-edk2.yaml:arch/v8.4.yaml
- PASS: build:ns-edk2-optee.yaml:arch/v8.4.yaml
- PASS: build:ffa-optee.yaml:arch/v8.4.yaml
- PASS: build:bootwrapper.yaml:arch/v8.4.yaml
- PASS: run:ns-preload.yaml:arch/v8.4.yaml
- PASS: run:ns-edk2.yaml:arch/v8.4.yaml:dt
- PASS: run:ns-edk2.yaml:arch/v8.4.yaml:acpi
- PASS: run:ns-edk2-optee.yaml:arch/v8.4.yaml
- PASS: run:ffa-optee.yaml:arch/v8.4.yaml:dt
- PASS: run:ffa-optee.yaml:arch/v8.4.yaml:acpi
- PASS: run:bootwrapper.yaml:arch/v8.4.yaml
- PASS: build:ns-preload.yaml:arch/v8.5.yaml
- PASS: build:ns-edk2.yaml:arch/v8.5.yaml
- PASS: build:ns-edk2-optee.yaml:arch/v8.5.yaml
- PASS: build:ffa-optee.yaml:arch/v8.5.yaml

- PASS: build:bootwrapper.yaml:arch/v8.5.yaml
- PASS: run:ns-preload.yaml:arch/v8.5.yaml
- PASS: run:ns-edk2.yaml:arch/v8.5.yaml:dt
- PASS: run:ns-edk2.yaml:arch/v8.5.yaml:acpi
- PASS: run:ns-edk2-optee.yaml:arch/v8.5.yaml
- PASS: run:ffa-optee.yaml:arch/v8.5.yaml:dt
- PASS: run:ffa-optee.yaml:arch/v8.5.yaml:acpi
- PASS: run:bootwrapper.yaml:arch/v8.5.yaml
- PASS: build:ns-preload.yaml:arch/v8.6.yaml
- PASS: build:ns-edk2.yaml:arch/v8.6.yaml
- PASS: build:ns-edk2-optee.yaml:arch/v8.6.yaml
- PASS: build:ffa-optee.yaml:arch/v8.6.yaml
- PASS: build:bootwrapper.yaml:arch/v8.6.yaml
- PASS: run:ns-preload.yaml:arch/v8.6.yaml
- PASS: run:ns-edk2.yaml:arch/v8.6.yaml:dt
- PASS: run:ns-edk2.yaml:arch/v8.6.yaml:acpi
- PASS: run:ns-edk2-optee.yaml:arch/v8.6.yaml
- PASS: run:ffa-optee.yaml:arch/v8.6.yaml:dt
- PASS: run:ffa-optee.yaml:arch/v8.6.yaml:acpi
- PASS: run:bootwrapper.yaml:arch/v8.6.yaml
- PASS: build:ns-preload.yaml:arch/v8.7.yaml
- PASS: build:ns-edk2.yaml:arch/v8.7.yaml
- PASS: build:ns-edk2-optee.yaml:arch/v8.7.yaml
- PASS: build:ffa-optee.yaml:arch/v8.7.yaml
- PASS: build:bootwrapper.yaml:arch/v8.7.yaml
- PASS: run:ns-preload.yaml:arch/v8.7.yaml
- PASS: run:ns-edk2.yaml:arch/v8.7.yaml:dt
- PASS: run:ns-edk2.yaml:arch/v8.7.yaml:acpi
- PASS: run:ns-edk2-optee.yaml:arch/v8.7.yaml
- PASS: run:ffa-optee.yaml:arch/v8.7.yaml:dt
- PASS: run:ffa-optee.yaml:arch/v8.7.yaml:acpi
- PASS: run:bootwrapper.yaml:arch/v8.7.yaml
- PASS: build:ns-preload.yaml:arch/v8.8.yaml
- PASS: build:ns-edk2.yaml:arch/v8.8.yaml
- PASS: build:ns-edk2-optee.yaml:arch/v8.8.yaml
- PASS: build:ffa-optee.yaml:arch/v8.8.yaml

- PASS: build:bootwrapper.yaml:arch/v8.8.yaml
- PASS: run:ns-preload.yaml:arch/v8.8.yaml
- PASS: run:ns-edk2.yaml:arch/v8.8.yaml:dt
- PASS: run:ns-edk2.yaml:arch/v8.8.yaml:acpi
- PASS: run:ns-edk2-optee.yaml:arch/v8.8.yaml
- PASS: run:ffa-optee.yaml:arch/v8.8.yaml:dt
- PASS: run:ffa-optee.yaml:arch/v8.8.yaml:acpi
- PASS: run:bootwrapper.yaml:arch/v8.8.yaml
- PASS: build:ns-preload.yaml:arch/v8.9.yaml
- PASS: build:ns-edk2.yaml:arch/v8.9.yaml
- PASS: build:ns-edk2-optee.yaml:arch/v8.9.yaml
- PASS: build:ffa-optee.yaml:arch/v8.9.yaml
- PASS: build:bootwrapper.yaml:arch/v8.9.yaml
- PASS: run:ns-preload.yaml:arch/v8.9.yaml
- PASS: run:ns-edk2.yaml:arch/v8.9.yaml:dt
- PASS: run:ns-edk2.yaml:arch/v8.9.yaml:acpi
- PASS: run:ns-edk2-optee.yaml:arch/v8.9.yaml
- PASS: run:ffa-optee.yaml:arch/v8.9.yaml:dt
- PASS: run:ffa-optee.yaml:arch/v8.9.yaml:acpi
- PASS: run:bootwrapper.yaml:arch/v8.9.yaml
- PASS: build:ns-preload.yaml:arch/v9.0.yaml
- PASS: build:ns-edk2.yaml:arch/v9.0.yaml
- PASS: build:ns-edk2-optee.yaml:arch/v9.0.yaml
- PASS: build:ffa-optee.yaml:arch/v9.0.yaml
- PASS: build:bootwrapper.yaml:arch/v9.0.yaml
- PASS: run:ns-preload.yaml:arch/v9.0.yaml
- PASS: run:ns-edk2.yaml:arch/v9.0.yaml:dt
- PASS: run:ns-edk2.yaml:arch/v9.0.yaml:acpi
- PASS: run:ns-edk2-optee.yaml:arch/v9.0.yaml
- PASS: run:ffa-optee.yaml:arch/v9.0.yaml:dt
- PASS: run:ffa-optee.yaml:arch/v9.0.yaml:acpi
- PASS: run:bootwrapper.yaml:arch/v9.0.yaml
- PASS: build:ns-preload.yaml:arch/v9.1.yaml
- PASS: build:ns-edk2.yaml:arch/v9.1.yaml
- PASS: build:ns-edk2-optee.yaml:arch/v9.1.yaml
- PASS: build:ffa-optee.yaml:arch/v9.1.yaml

- PASS: build:bootwrapper.yaml:arch/v9.1.yaml
- PASS: run:ns-preload.yaml:arch/v9.1.yaml
- PASS: run:ns-edk2.yaml:arch/v9.1.yaml:dt
- PASS: run:ns-edk2.yaml:arch/v9.1.yaml:acpi
- PASS: run:ns-edk2-optee.yaml:arch/v9.1.yaml
- PASS: run:ffa-optee.yaml:arch/v9.1.yaml:dt
- PASS: run:ffa-optee.yaml:arch/v9.1.yaml:acpi
- PASS: run:bootwrapper.yaml:arch/v9.1.yaml
- PASS: build:ns-preload.yaml:arch/v9.2.yaml
- PASS: build:ns-edk2.yaml:arch/v9.2.yaml
- PASS: build:ns-edk2-optee.yaml:arch/v9.2.yaml
- PASS: build:ffa-optee.yaml:arch/v9.2.yaml
- PASS: build:bootwrapper.yaml:arch/v9.2.yaml
- PASS: run:ns-preload.yaml:arch/v9.2.yaml
- PASS: run:ns-edk2.yaml:arch/v9.2.yaml:dt
- PASS: run:ns-edk2.yaml:arch/v9.2.yaml:acpi
- PASS: run:ns-edk2-optee.yaml:arch/v9.2.yaml
- PASS: run:ffa-optee.yaml:arch/v9.2.yaml:dt
- PASS: run:ffa-optee.yaml:arch/v9.2.yaml:acpi
- PASS: run:bootwrapper.yaml:arch/v9.2.yaml
- PASS: build:ns-preload.yaml:arch/v9.3.yaml
- PASS: build:ns-edk2.yaml:arch/v9.3.yaml
- PASS: build:ns-edk2-optee.yaml:arch/v9.3.yaml
- PASS: build:ffa-optee.yaml:arch/v9.3.yaml
- PASS: build:bootwrapper.yaml:arch/v9.3.yaml
- PASS: run:ns-preload.yaml:arch/v9.3.yaml
- PASS: run:ns-edk2.yaml:arch/v9.3.yaml:dt
- PASS: run:ns-edk2.yaml:arch/v9.3.yaml:acpi
- PASS: run:ns-edk2-optee.yaml:arch/v9.3.yaml
- PASS: run:ffa-optee.yaml:arch/v9.3.yaml:dt
- PASS: run:ffa-optee.yaml:arch/v9.3.yaml:acpi
- PASS: run:bootwrapper.yaml:arch/v9.3.yaml
- PASS: build:ns-preload.yaml:arch/v9.4.yaml
- PASS: build:ns-edk2.yaml:arch/v9.4.yaml
- PASS: build:ns-edk2-optee.yaml:arch/v9.4.yaml
- PASS: build:ffa-optee.yaml:arch/v9.4.yaml

- PASS: run:ns-preload.yaml:arch/v9.4.yaml
- PASS: run:ns-edk2.yaml:arch/v9.4.yaml:dt
- PASS: run:ns-edk2.yaml:arch/v9.4.yaml:acpi
- PASS: run:ns-edk2-optee.yaml:arch/v9.4.yaml
- PASS: run:ffa-optee.yaml:arch/v9.4.yaml:dt
- PASS: run:ffa-optee.yaml:arch/v9.4.yaml:acpi
- PASS: build:ns-preload.yaml:arch/v9.5.yaml
- PASS: build:ns-edk2.yaml:arch/v9.5.yaml
- PASS: build:ns-edk2-optee.yaml:arch/v9.5.yaml
- PASS: build:ffa-optee.yaml:arch/v9.5.yaml
- PASS: run:ns-preload.yaml:arch/v9.5.yaml
- PASS: run:ns-edk2.yaml:arch/v9.5.yaml:dt
- PASS: run:ns-edk2.yaml:arch/v9.5.yaml:acpi
- PASS: run:ns-edk2-optee.yaml:arch/v9.5.yaml
- PASS: run:ffa-optee.yaml:arch/v9.5.yaml:dt
- PASS: run:ffa-optee.yaml:arch/v9.5.yaml:acpi
- PASS: build:ffa-hafnium-optee.yaml:arch/v8.5.yaml
- PASS: build:ffa-tftf.yaml:arch/v8.5.yaml
- PASS: run:ffa-hafnium-optee.yaml:arch/v8.5.yaml:dt
- PASS: run:ffa-hafnium-optee.yaml:arch/v8.5.yaml:acpi
- PASS: run:ffa-tftf.yaml:arch/v8.5.yaml:dt
- PASS: run:ffa-tftf.yaml:arch/v8.5.yaml:acpi
- PASS: build:ffa-hafnium-optee.yaml:arch/v8.6.yaml
- PASS: build:ffa-tftf.yaml:arch/v8.6.yaml
- PASS: run:ffa-hafnium-optee.yaml:arch/v8.6.yaml:dt
- PASS: run:ffa-hafnium-optee.yaml:arch/v8.6.yaml:acpi
- PASS: run:ffa-tftf.yaml:arch/v8.6.yaml:dt
- PASS: run:ffa-tftf.yaml:arch/v8.6.yaml:acpi
- PASS: build:ffa-hafnium-optee.yaml:arch/v8.7.yaml
- PASS: build:ffa-tftf.yaml:arch/v8.7.yaml
- PASS: run:ffa-hafnium-optee.yaml:arch/v8.7.yaml:dt
- PASS: run:ffa-hafnium-optee.yaml:arch/v8.7.yaml:acpi
- PASS: run:ffa-tftf.yaml:arch/v8.7.yaml:dt
- PASS: run:ffa-tftf.yaml:arch/v8.7.yaml:acpi
- PASS: build:ffa-hafnium-optee.yaml:arch/v8.8.yaml
- PASS: build:ffa-tftf.yaml:arch/v8.8.yaml

- PASS: run:ffa-hafnium-optee.yaml:arch/v8.8.yaml:dt
- PASS: run:ffa-hafnium-optee.yaml:arch/v8.8.yaml:acpi
- PASS: run:ffa-tftf.yaml:arch/v8.8.yaml:dt
- PASS: run:ffa-tftf.yaml:arch/v8.8.yaml:acpi
- PASS: build:ffa-hafnium-optee.yaml:arch/v8.9.yaml
- PASS: build:ffa-tftf.yaml:arch/v8.9.yaml
- PASS: run:ffa-hafnium-optee.yaml:arch/v8.9.yaml:dt
- PASS: run:ffa-hafnium-optee.yaml:arch/v8.9.yaml:acpi
- PASS: run:ffa-tftf.yaml:arch/v8.9.yaml:dt
- PASS: run:ffa-tftf.yaml:arch/v8.9.yaml:acpi
- PASS: build:ffa-hafnium-optee.yaml:arch/v9.0.yaml
- PASS: build:ffa-tftf.yaml:arch/v9.0.yaml
- PASS: run:ffa-hafnium-optee.yaml:arch/v9.0.yaml:dt
- PASS: run:ffa-hafnium-optee.yaml:arch/v9.0.yaml:acpi
- PASS: run:ffa-tftf.yaml:arch/v9.0.yaml:dt
- PASS: run:ffa-tftf.yaml:arch/v9.0.yaml:acpi
- PASS: build:ffa-hafnium-optee.yaml:arch/v9.1.yaml
- PASS: build:ffa-tftf.yaml:arch/v9.1.yaml
- PASS: run:ffa-hafnium-optee.yaml:arch/v9.1.yaml:dt
- PASS: run:ffa-hafnium-optee.yaml:arch/v9.1.yaml:acpi
- PASS: run:ffa-tftf.yaml:arch/v9.1.yaml:dt
- PASS: run:ffa-tftf.yaml:arch/v9.1.yaml:acpi
- PASS: build:ffa-hafnium-optee.yaml:arch/v9.2.yaml
- PASS: build:ffa-tftf.yaml:arch/v9.2.yaml
- PASS: run:ffa-hafnium-optee.yaml:arch/v9.2.yaml:dt
- PASS: run:ffa-hafnium-optee.yaml:arch/v9.2.yaml:acpi
- PASS: run:ffa-tftf.yaml:arch/v9.2.yaml:dt
- PASS: run:ffa-tftf.yaml:arch/v9.2.yaml:acpi
- PASS: build:ffa-hafnium-optee.yaml:arch/v9.3.yaml
- PASS: build:ffa-tftf.yaml:arch/v9.3.yaml
- PASS: run:ffa-hafnium-optee.yaml:arch/v9.3.yaml:dt
- PASS: run:ffa-hafnium-optee.yaml:arch/v9.3.yaml:acpi
- PASS: run:ffa-tftf.yaml:arch/v9.3.yaml:dt
- PASS: run:ffa-tftf.yaml:arch/v9.3.yaml:acpi
- PASS: build:ffa-hafnium-optee.yaml:arch/v9.4.yaml
- PASS: build:ffa-tftf.yaml:arch/v9.4.yaml

- PASS: run:ffa-hafnium-optee.yaml:arch/v9.4.yaml:dt
- PASS: run:ffa-hafnium-optee.yaml:arch/v9.4.yaml:acpi
- PASS: run:ffa-tftf.yaml:arch/v9.4.yaml:dt
- PASS: run:ffa-tftf.yaml:arch/v9.4.yaml:acpi
- PASS: build:ffa-hafnium-optee.yaml:arch/v9.5.yaml
- PASS: build:ffa-tftf.yaml:arch/v9.5.yaml
- PASS: run:ffa-hafnium-optee.yaml:arch/v9.5.yaml:dt
- PASS: run:ffa-hafnium-optee.yaml:arch/v9.5.yaml:acpi
- PASS: run:ffa-tftf.yaml:arch/v9.5.yaml:dt
- PASS: run:ffa-tftf.yaml:arch/v9.5.yaml:acpi
- PASS: build:cca-3world.yaml
- PASS: build:cca-edk2.yaml
- PASS: build:rfa.yaml
- PASS: run:cca-3world.yaml
- PASS: run:cca-edk2.yaml:dt
- PASS: run:cca-edk2.yaml:acpi
- PASS: run:rfa.yaml
- PASS: build:cca-3world.yaml:buildroot-cca.yaml,test/cca.yaml
- PASS: run:cca-3world.yaml:buildroot-cca.yaml,test/cca.yaml:realm
- PASS: build:cca-4world.yaml
- PASS: run:cca-4world.yaml
- PASS: repo-sync-behaviours

1.5 License

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

(continues on next page)

(continued from previous page)

```
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.
```

1.5.1 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/