Ventana SDK User Manual

Rev 0.18.1

Jun 30, 2025

Table of contents

1. Introduction

The Ventana Software SDK provides a complete set of tools and software to build and run applications for the Veyron family of processors.

This document describes how to download the Ventana Software Release components, including pre-built binaries, and steps to boot Linux on supported platforms. The document also describes steps to get the source code from GitHub and build individual components of the Ventana Software Release.

1.1. Supported Platforms

Platform	Description
Synth V2	QEMU Emulated Platform for Ventana Veyron V2
Synth E2	QEMU Emulated Platform for Ventana Veyron E2
Thunderhill V2	Ventana Veyron V2 FPGA Platform

1.2. SDK Contents

Category	Component	Description	Contents
Firmware	zStage	Zero stage initial boot-loader	Binaries and sources provided
BIOS	OpenSBI	Runtime	Binaries and sources provided
	EDK II UEFI	Boot Loader	

	U-Boot	Boot Loader	
Kernel	Linux	Kernel	Binaries and sources provided
Compiler & Toolchain	GCC	Native and Cross-compilation toolchains	Binaries and sources provided
	LLVM		
Emulation Platform	QEMU Synth	Emulation Platform	Sources provided

2. Downloads

Download the necessary components to a common directory. The rest of this document will refer to this directory as "work directory" or `WORK_DIR`.

- Pre-built binaries
  - zStage
  - OpenSBI
  - U-Boot
  - EDK II
  - Linux
  - ventana-sw-minimal-apps_${VENTANA_SW_VERSION}.tar.xz for Ventana applications

    https://drive.google.com/drive/folders/1ziM6oW0G4EVlQPrvyhffiE2CZ6GD2XPo?usp=drive_link

- Ventana toolchains
  - GCC – Ventana native toolchain
  - GCC – Ventana cross toolchain
  - LLVM - Ventana native toolchain
  - LLVM – Ventana cross toolchain

- Release notes

- Source code
  Source code is not required if pre-built binaries are utilized. Source code is required if components are to be rebuilt.
  - QEMU
    ```
    $ git clone -b release-v0.18.1
    git@github.com:ventanamicro-dev/qemu
    ```
  - zStage
    ```
    $ git clone -b release-v0.18.1
    git@github.com:ventanamicro-dev/zstage
    ```
  - OpenSBI
    ```
    $ git clone -b release-v0.18.1
    git@github.com:ventanamicro-dev/opensbi
    ```
  - EDK II
    ```
    $ git clone -b release-v0.18.1
    git@github.com:ventanamicro-dev/edk2-platforms
    ```
  - U-Boot
    ```
    $ git clone -b release-v0.18.1
    git@github.com:ventanamicro-dev/u-boot
    ```
  - Linux
    ```
    $ git clone -b release-v0.18.1
    git@github.com:ventanamicro-dev/linux
    ```

3. Booting and Running

3.1. QEMU

An open source project called "QEMU" provides the capability to emulate a complete system, booting an operating system running on an emulated processor architecture. An emulated system running on Ventana Veyron processors can be used to evaluate the capabilities and functionality of this technology, as well as facilitate software development.

3.1.1. System Requirements

For running QEMU, the following system capabilities are recommended at a minimum:

- 16 GB memory
- 200 GB storage
- 8 core CPU
- Ubuntu 22.04

3.1.2. Ventana pre-built binaries and Ubuntu pre-installed images

Pre-built binaries are available on Ventana Google Drive. Refer to Build individual components with Ventana source code to build Ventana binaries.

Unpack the pre-built binaries of zStage, OpenSBI, EDK II platform, U-Boot, Linux kernel, for QEMU for Supported platforms into `$WORK_DIR/ventana-sw` directory.

```
$  export VENTANA_SW_VERSION=0.18.1
$  export WORK_DIR=$(pwd)
$  cd $WORK_DIR
$  mkdir ventana-sw
$  tar -C ventana-sw -xf ventana-sw-${VENTANA_SW_VERSION}.tar.xz
$  cd $WORK_DIR/ventana-sw
```

Download and prepare Ubuntu pre-installed disk image as follows:

1. Download and extract ubuntu 25.04 pre-installed image from Canonical website:

```
$  wget
   https://cdimage.ubuntu.com/releases/25.04/release/ubuntu-25.04-pre
   installed-server-riscv64.img.xz

$  xz -dv ubuntu-25.04-preinstalled-server-riscv64.img.xz
```

2. On a separate terminal go to `$WORK_DIR/ventana-sw` directory

```
$  cd $WORK_DIR/ventana-sw
```

3. Run a temporary web-server in `$WORK_DIR/ventana-sw` directory which will allow cloud-init to download Ventana packages

```
$  python3 -m http.server --directory $WORK_DIR/ventana-sw 8000
```

4. Add additional space to disk image before booting on QEMU:

```
$  qemu-img resize -f raw
   ubuntu-25.04-preinstalled-server-riscv64.img +<extra_gb_count>G
```

   NOTE: Replace `<extra_gb_count>` with the desired number of additional gigabytes of storage. Our recommendation is to use at least `15` as `<extra_gb_count>`.

   NOTE: To configure Ubuntu pre-installed disk image with Ventana native toolchain use `ventana-cloud-ubuntu-toolchain.iso` instead of `ventana-cloud-ubuntu.iso` and download `ventana-native-toolchain-2025.05.22.tar.xz` in `$WORK_DIR/ventana-sw`

5. Configure the Ubuntu-25.04 pre-installed image using ventana-cloud-ubuntu.iso

```
$  ./QEMU-x86_64-Ubuntu-20.04.AppImage -M ventana-synth-v2 -nographic \
   -bios ./platform/ventana-synth-v2/zstage/zstage.bin \
   -sd ./platform/bios_edk2_riscv64_ventana.img \
   -device virtio-net-pci,netdev=eth0 -netdev
   user,hostfwd=tcp::9991-:22,id=eth0 \
   -drive file=./ventana-cloud-ubuntu.iso,id=hd1,format=raw \
   -device nvme,serial=aaddaadd,drive=hd1 \
```

```
    -drive
    file=./ubuntu-25.04-preinstalled-server-riscv64.img,id=hd0,format=ra
    w \
    -device nvme,serial=ddaaddaa,drive=hd0
```

NOTE: Cloud init based configuration works in background so no need to log into the system. Also, after configuration is complete the system will poweroff automatically.

6. Terminate the temporary web-server started in step 3 by pressing <ctrl>-C in that terminal session.

7.

```
jenkins@yocto-PowerEdge-T440:~/ventana-sw/0.18.1$ ./QEMU-x86_64-Ubuntu-20.04.AppImage -M ventana-synth-v2 -nographic \
-bios ./platform/ventana-synth-v2/zstage/zstage.bin \
-sd ./platform/bios_edk2_riscv64_ventana.img \
-device virtio-net-pci,netdev=eth0 -netdev user,hostfwd=tcp::9991-:22,id=eth0 \
-drive file=./ventana-cloud-ubuntu.iso,id=hd1,format=raw \
-device nvme,serial=aaddaadd,drive=hd1 \
-drive file=./ubuntu-25.04-preinstalled-server-riscv64.img,id=hd0,format=raw \
-device nvme,serial=ddaaddaa,drive=hd0
WARNING: Image format was not specified for './platform/bios_edk2_riscv64_ventana.img' and probing guessed raw.
         Automatically detecting the format is dangerous for raw images, write operations on block 0 will be restricted.
         Specify the 'raw' format explicitly to remove the restrictions.
QEMU-x86_64-Ubuntu-20.04.AppImage: info: Unsigned image: Secure Boot is disabled for zStage

Ventana zStage v0.1
zstage: console device uart8250
zstage: setup platform FDT at 0x83f00000
zstage: copying next stage (135056 bytes) to 0x84000000 from 0x80021000
zstage: next booting stage at 0x84000000 (135056 bytes)
zstage: harts 0x3*, 0x1, 0x2, 0x0
Updating cache-size from 134217728 to 16777216
Updating cache-sets from 131072 to 16384
zstage: Unsigned image: Secure Boot is disabled for next booting stage
zstage: jumping to next booting stage

U-Boot SPL 2025.07-rc3-00018-gf870bb5109c9 (May 30 2025 - 12:49:48 +0530)
Trying to boot from MMC1
## Checking signature(s) for config 'conf-0' ... ==========> OK
## Checking signature(s) and/or hash(es) for Image 'opensbi' ... sha512+ sha384,rsa3072:ventana_rsa-1- ===> OK
## Checking signature(s) and/or hash(es) for Image 'edk2' ... sha512+ sha384,rsa3072:ventana_rsa-2- ===> OK
WARNING: riscv,rpmi-system-reset driver is experimental and may change
WARNING: riscv,rpmi-shmem-mbox driver is experimental and may change
WARNING: riscv,rpmi-system-suspend driver is experimental and may change
WARNING: riscv,rpmi-hsm driver is experimental and may change
WARNING: riscv,rpmi-cppc driver is experimental and may change

OpenSBI v1.6-130-g02690e2e
   ____                    _____ ____ _____
  / __ \                  / ____|  _ \_   _|
 | |  | |_ __   ___ _ __ | (___ | |_) || |
 | |  | | '_ \ / _ \ '_ \ \___ \|  _ < | |
 | |__| | |_) |  __/ | | |____) | |_) || |_
  \____/| .__/ \___|_| |_|_____/|____/_____|
        | |
        |_|

WARNING: riscv,rpmi-mpxy-system-msi driver is experimental and may change
WARNING: riscv,rpmi-mpxy-clock driver is experimental and may change
Platform Name               : Ventana Synthetic v2 Board
Platform Features           : medeleg
Platform HART Count         : 4
Platform IPI Device         : aia-imsic
Platform Timer Device       : aclint-mtimer @ 100000000Hz
Platform Console Device     : uart8250
Platform HSM Device         : rpmi-hsm
Platform PMU Device         : ---
Platform Reboot Device      : rpmi-system-reset
Platform Shutdown Device    : rpmi-system-reset
Platform Suspend Device     : rpmi-system-suspend
Platform CPPC Device        : rpmi-cppc
Platform RAS Device         : sbi-ras-agent
Firmware Base               : 0x87000000
Firmware Size               : 353 KB
Firmware RW Offset          : 0x40000
Firmware RW Size            : 97 KB
Firmware Heap Offset        : 0x4c000
Firmware Heap Size          : 49 KB (total), 3 KB (reserved), 19 KB (used), 26 KB (free)
Firmware Scratch Size       : 4096 B (total), 1528 B (used), 2568 B (free)
Runtime SBI Version         : 3.0
Standard SBI Extensions     : time,rfnc,ipi,base,hsm,srst,susp,pmu,dbcn,cppc,fwft,legacy,dbtr,sse,mpxy
Experimental SBI Extensions : none

Domain0 Name                : root
Domain0 Boot HART           : 3
```

NOTE: The `ventana-cloud-ubuntu.iso` configures Ubuntu pre-installed disk image without Ventana native toolchain.

3.1.3. Run images on QEMU platform

3.1.3.1. QEMU options

QEMU Options	Values	Description
`-M`	`ventana-synth-v2` `ventana-synth-e2`	Platform
`-bios`	`/path/to/zstage.bin`	Zero-stage loader
`-netdev`	`user,id=net0`	Network interface – backend
`-device`	`nvme,serial=ddaaddaa,drive=hd0`	Disk drive – frontend device
`-smp <n>`	`Up to 32 CPUs` `(default: minimum of [32, number of host CPUs])`	Number of CPUs
`-m`	`Minimum 4GB (default 4GB)`	Memory

Table 1: QEMU options for the Ventana Synth platforms

3.1.3.2. Boot QEMU with Ventana Synth V2

Run an emulated 64-bit Ventana Synth V2 machine using QEMU with the following command:

```
$  ./QEMU-x86_64-Ubuntu-20.04.AppImage -M ventana-synth-v2 -nographic \
   -bios ./platform/ventana-synth-v2/zstage/zstage.bin \
   -sd ./platform/bios_edk2_riscv64_ventana.img \
   -device virtio-net-pci,netdev=eth0 -netdev
   user,hostfwd=tcp::9991-:22,id=eth0 \
```

```
   -drive
   file=./ubuntu-25.04-preinstalled-server-riscv64.img,id=hd0,format=raw \
   -device nvme,serial=ddaaddaa,drive=hd0
```

NOTE: The above QEMU command will create a QEMU ventana-synth-v2 machine with the number of CPUs matching the number of CPUs on the host system, up to 32 CPUs. The ventana-synth-v2 platform supports up to 32 CPUs. To specify a particular number "<n>" of CPUs, add "`-smp <n>`" parameter to the above QEMU command.

To poweroff QEMU:

```
$  sudo poweroff
```

3.1.3.3. Boot QEMU with Ventana Synth E2

Run an emulated 64-bit Ventana Synth E2 machine using QEMU with the following command:

```
$  ./QEMU-x86_64-Ubuntu-20.04.AppImage -M ventana-synth-e2 -nographic \
   -bios ./platform/ventana-synth-e2/zstage/zstage.bin \
   -sd ./platform/bios_edk2_riscv64_ventana.img \
   -device virtio-net-pci,netdev=eth0 -netdev
   user,hostfwd=tcp::9991-:22,id=eth0 \
   -drive
   file=./ubuntu-25.04-preinstalled-server-riscv64.img,id=hd0,format=raw \
   -device nvme,serial=ddaaddaa,drive=hd0
```

NOTE: The above QEMU command will create a QEMU ventana-synth-e2 machine with the number of CPUs matching the number of CPUs on the host system, up to 32 CPUs. The ventana-synth-e2 platform supports up to 32 CPUs. To specify a particular number "<n>" of CPUs, add "`-smp <n>`" parameter to the above QEMU command.

To poweroff QEMU:

```
$  sudo poweroff
```

3.2. Thunderhill V2 Platform

The Thunderhill V2 is a feature rich FPGA platform for evaluating Ventana Veyron V2 and other Ventana IPs. For more information, please refer to the Ventana Veyron V2 FPGA user guide.

The Thunderhill V2 platform supports booting from SD Card and a NVMe SSD or SATA disk where the SD card is also the BIOS boot device.

3.2.2. Prepare the SD Card

- Plug the SD card into the host system and consult the dmesg output to check the device name for the SD card. Some examples are /dev/mmcblock0, /dev/sdc etc.
- Use the command below to flash the image created above into the SD card:

```
$  dd if=/path/to/bios_edk2_riscv64_ventana.img of=/dev/<device name>
```

3.2.3. NVMe Disk Booting

To boot from an NVMe SSD disk is required along with the SD card which has the required firmware image flashed as described in the step 3.2.2. Following is a recommended product for the NVMe SSD M.2 drive:
https://www.crucial.in/products/ssd/crucial-p5-plus-ssd

3.2.3.1. Prepare NVMe Disk

- Plug the NVMe disk into the host system and consult the dmesg output to check the device name for the NVMe disk. Some examples are /dev/nvme0, /dev/nvme1 etc.
- Use the command below to flash the raw image (created in step 3.2.1 above) into the NVMe disk:

```
$  dd if=/path/to/ubuntu-25.04-preinstalled-server-riscv64.img
   of=/dev/<device name>
```

3.2.4. SATA Disk Booting

To boot from a SATA disk a PCIe adapter and SATA SSD are required along with an SD card which has the required firmware image flashed as described in step 3.2.2. Following are recommended products for SATA controller and drives:

https://www.amazon.com/dp/B09KCH846V?psc=1&ref=ppx_yo2ov_dt_b_product_details
https://www.amazon.com/dp/B07YD579WM?psc=1&ref=ppx_yo2ov_dt_b_product_details

3.2.4.1. Prepare SATA Disk

- Plug the SATA disk into the host system and consult the dmesg output to check the device name for the SATA disk. Some examples are /dev/sda, /dev/sdb etc.
- Use the command below to flash the raw image (created in step 3.2.1 above) into the SATA disk:

```
$  dd if=/path/to/
   ubuntu-25.04-preinstalled-server-riscv64+ventana.raw
   of=/dev/<device name>
```

3.2.5. Booting from NVMe/SATA Disk

Now plug the SD card and the NVMe/SATA disk into the board and the M.2 slot/PCIe adapter respectively and power it on. Once the firmware boots up it should start booting from the SD card and then boot the operating system from the connected NVMe/SATA disk.

4. Compiling for Ventana Veyron processors

This chapter describes how to build libraries and applications for running on the Veyron family of processors. Two methods are described:

- Cross Compile (building RISC-V binaries on an x86 system)
- Native Compile (building RISC-V binaries on a RISC-V system)

4.1. Installing the Ventana cross toolchain

The `ventana-cross-toolchain` package provides pre-built GCC and LLVM cross toolchains that run on x86 systems and produce binaries for the Ventana Veyron family of processors.

In order to use a pre-built Ventana cross toolchain, download and extract the `ventana-cross-toolchain` package on the host machine:

```
$  cd $WORK_DIR
$  tar -xf ventana-cross-toolchain-2025.05.22.tar.xz
```

4.2. Preparing to use the Ventana cross toolchain

Set the following environment variables to begin using the Ventana cross toolchain:

```
$  export ARCH=riscv
$  export CROSS_COMPILE=riscv64-unknown-linux-gnu-
$  export PATH=${WORK_DIR}/ventana-cross-toolchain-2025.05.22/bin:${PATH}
```

4.3. Cross-compiling an application

EEMBC's CoreMark® is an interesting benchmark often used for testing the performance of processor cores. Building and running this benchmark can be used to demonstrate cross-compilation of an application.

1. Install the toolchain, as described in Installing the Ventana cross toolchain.
2. Set up environment to use the toolchain, as described in Preparing to use the Ventana cross toolchain
3. Download the source onto the system on which the cross toolchain is installed:
   ```
   $  git clone https://github.com/eembc/coremark.git
   ```
4. Build with GCC compiler:
   ```
   $  cd coremark
   ```

```
$  make CC=${CROSS_COMPILE}gcc link
```

Or, to build with LLVM compiler:

```
$  make CC=clang link
```

The resulting executable is `coremark.exe`.

5. Boot QEMU (See Booting and running).
6. On host machine, copy the compiled executable file to QEMU guest:

```
$ scp -P 9991 coremark.exe ventana@localhost:
```

In QEMU guest, executable file will be copied to "/home/root":

```
ventana@ventana:~# ls -l
total 36
-rwxr-xr-x 1 root root 12552 Dec  9 08:47 coremark.exe
```

7. To execute the newly built command, from the QEMU guest console:

```
ventana@ventana:~# ./coremark.exe
2K performance run parameters for coremark.
[...]
```

4.4. Installing the Ventana native toolchain

The `ventana-native-toolchain` package provides pre-built GCC and LLVM native toolchains that run on RISC-V platforms and produce binaries for the Ventana Veyron family of processors.

4.5. Using the Ventana native toolchain

After logging into a system using a provided Ubuntu 25.04 root filesystem, the default user's `PATH` is already modified to use the Ventana native toolchain by default.

4.6. Compiling a real-world example

1. Download the source onto the system on which the native toolchain is installed:

```
ventana:~$ git clone https://github.com/eembc/coremark.git
```

2. Build with GCC compiler:

```
ventana:~$ cd coremark
ventana:~$ make CC=gcc link
```

Or, to build with LLVM compiler:

```
ventana:~$ make CC=clang link
```

3. Run the application:

```
ventana:~$ ./coremark.exe
```

5. Build individual components with Ventana source code

The below subsections provide steps to download source code and build individual components. For building components of the boot stack, the following system capabilities are recommended at a minimum:

- 16 GB memory
- 200 GB storage
- 8 core CPU
- Ubuntu 22.04

5.1. OpenSBI image

Table 2 below lists the OpenSBI build input components and their paths. If one or more of these components are modified, the OpenSBI image needs to be rebuilt. Table 3 lists the commands for building OpenSBI.

OpenSBI build input components	**Download command**`git clone -b release-v0.18.1 \ git@github.com:ventanamicro-dev/opensbi.git`
OpenSBI source	$

Table 2 : Opensbi build components

Build objective	**Build commands**

Build OpenSBI	`$ make -C opensbi clean distclean` `$ make -C opensbi PLATFORM=generic FW_OPTIONS=0x0`

Table 3 : OpenSBI build commands

5.2. Linux kernel image

Table 4 below lists the kernel build input components and their paths. If one or more of these components are modified the kernel image needs to be rebuilt. Table 5 lists the several commands available for building the kernel.

Kernel build input components	**Component path**
Kernel source	`$ git clone -b release-v0.18.1 git@github.com:ventanamicro-dev/linux.git`
Kernel defconfigs	• `linux/arch/riscv/configs/ventana_sw_defconfig` • `linux/arch/riscv/configs/ventana_sw_minimal_defconfig`

Table 4 : kernel build components

Note: Use `ventana_sw_minimal_defconfig` for constrained environments, such as the Ventana HAPS offering.

Build objective	**Build commands**

Build the kernel	`$ make -C linux distclean`
	`$ make -C linux ventana_sw_defconfig`
	[Or, `ventana_sw_minimal_defconfig`]
	`$ make -C linux DPKG_FLAGS="-d" deb-pkg LOCALVERSION=-ventana`

Table 5 : kernel build commands

5.3. U-Boot image

Table 6 below lists the U-Boot build input components and their paths. If one or more of these components are modified the U-Boot image needs to be rebuilt. Table 7 lists the several commands available for building U-Boot.

U-Boot build input components	**Component path**
U-Boot source	`$ git clone -b release-v0.18.1 \` `git@github.com:ventanamicro-dev/u-boot.git`
Ventana Synth V2 machine U-boot default configuration file	`u-boot/configs/ventana-vtx-synth_defconfig`
VentanaThunderhill V2 machine U-boot default configuration file	`u-boot/configs/ventana-vtx-thunderhill_defconfig`
Dependency	OpenSBI

Table 6 : U-Boot build components

Build objective	**Build commands**

Build U-boot for Ventana Synth V2	`$ make -C u-boot distclean`
	`$ make -C u-boot ventana-vtx-synth_defconfig`
	`$ make -C u-boot \`
	`OPENSBI=${WORK_DIR}/opensbi/build/platform/generic/firmware/fw_dynamic.bin`
	
	Note: fw_dynamic.bin is OpenSBI firmware image. To build OpenSBI firmware image, refer to OpenSBI image

Build U-boot for Ventana Thunderhill V2	`$ make -C u-boot distclean`
	`$ make -C u-boot ventana-vtx-thunderhill_defconfig`
	`$ make -C u-boot \`

	`OPENSBI=${WORK_DIR}/opensbi/build/platform/generic/firmware/fw_dynamic.bin` Note: fw_dynamic.bin is OpenSBI firmware image. To build OpenSBI firmware image, refer to OpenSBI image

Table 7 : U-Boot build commands

5.4. zStage image

Table 8 below lists the zStage build input components and their paths. If one or more of these components are modified, the zStage image needs to be rebuilt. Table 9 shows the commands to build zStage image.

zStage build input components	Component path
zStage source	`$ git clone -b release-v0.18.1 \` `git@github.com:ventanamicro-dev/zstage.git`
Dependency	U-Boot

Table 8 : zStage build components

Build objective	Build commands

Build zStage for Ventana Synth V2	`$ make -C zstage clean distclean`
	`$ make -C zstage PLATFORM=ventana/vt2/synth \`
	`ZSTAGE_PAYLOAD_PATH=${WORK_DIR}/u-boot/spl/u-boot-spl.bin`

Build zStage for Ventana Synth E2	`$ make -C zstage clean distclean`
	`$ make -C zstage PLATFORM=ventana/vt2/synth \`
	`VT2_SYNTH_E2=1 \`
	`ZSTAGE_PAYLOAD_PATH=${WORK_DIR}/u-boot/spl/u-boot-spl.bin`

Build zStage for Ventana Thunderhill V2	`$ make -C zstage clean distclean`
	`$ make -C zstage PLATFORM=ventana/vt2/thunderhill \`
	`ZSTAGE_PAYLOAD_PATH=${WORK_DIR}/u-boot/spl/u-boot-spl.bin`

Table 9 : zStage build commands

5.5. EDK II image

Table 10 below lists the EDK II build input components and their paths. If one or more of these components are modified, the EDK II image needs to be rebuilt. Table 11 shows the commands to build EDK II image.

EDK II build input components	Download command
EDK II source	`$ git clone -b release-v0.18.1 --recurse-submodule \` `git@github.com:ventanamicro-dev/edk2.git`

EDK II firmware source	`$ cd edk2`
	`$ git clone -b release-v0.18.1 --recurse-submodule \`
	`git@github.com:ventanamicro-dev/edk2-platforms.git`
	`$ cd ../`

Table 10 : EDK2 build components

Build objective	Build commands

Build EDK II firmware for Ventana Synth V2	`$ export WORKSPACE=`pwd``
	`$ export`
	`GCC5_RISCV64_PREFIX=$WORK_DIR/ventana-cross-toolchain-2025.05.22/bin/riscv64-unknown-linux-gnu-`
	`$ export \`
	`PACKAGES_PATH=$WORKSPACE/edk2:$WORKSPACE/edk2/edk2-platforms`
	`$ export EDK_TOOLS_PATH=$WORKSPACE/edk2/BaseTools`
	`$ source edk2/edksetup.sh --reconfig`
	`$ make -C edk2/BaseTools clean`
	`$ make -C edk2/BaseTools`
	`$ make -C edk2/BaseTools/Source/C`
	`$ source edk2/edksetup.sh BaseTools`
	`$ build -a RISCV64 --buildtarget RELEASE \`
	`-D FIRMWARE_VER="stable202505" \`
	`-p`
	`edk2/edk2-platforms/Platform/VentanaMicro/VentanaSynth/VentanaSynth.dsc \`
	`-t GCC5`

Build edk2-fit.itb for Ventana Synth V2	Prerequisite: u-boot-tools should be installed. Update the paths in ${WORK_DIR}/<path to DTS file>/ventana-edk2-synth.dts to point to OpenSBI (build/platform/generic/firmware/fw_dynamic.bin) and EDK II (Build/VentanaSynth/RELEASE_GCC5/FV/VENTANASYNTH.fd) images instead of prebuilt binaries. `$ mkimage -f \ ${WORK_DIR}/<path to DTS file>/ventana-edk2-synth.dts ventana-synth-edk2-fit.itb` **For information about FIT images and creating DTS files see FIT images.**

Build EDK II firmware for Ventana Thunderhill V2	`$ export WORKSPACE=`pwd``
	`$ export GCC5_RISCV64_PREFIX=$WORK_DIR/ventana-cross-toolchain-2025.05.22/bin/riscv64-unknown-linux-gnu-`
	`$ export \`
	`PACKAGES_PATH=$WORKSPACE/edk2:$WORKSPACE/edk2/edk2-platforms`
	`$ export EDK_TOOLS_PATH=$WORKSPACE/edk2/BaseTools`
	`$ source edk2/edksetup.sh --reconfig`
	`$ make -C edk2/BaseTools clean`
	`$ make -C edk2/BaseTools`
	`$ make -C edk2/BaseTools/Source/C`
	`$ source edk2/edksetup.sh BaseTools`
	`$ build -a RISCV64 --buildtarget RELEASE \`
	`-D FIRMWARE_VER="stable202505" \`
	`-p edk2/edk2-platforms/Platform/VentanaMicro/Thunderhill/Thunderhill.dsc \`
	`-t GCC5`

Build edk2-fit.itb for Ventana Thunderhill V2	Prerequisite: u-boot-tools should be installed. Update the paths in ${WORK_DIR}/<path to DTS file>/ventana-edk2-thunderhill.dts to point to OpenSBI (build/platform/generic/firmware/fw_dynamic.bin) and EDK II (Build/Thunderhill/RELEASE_GCC5/FV/THUNDERHILL.fd) images instead of prebuilt binaries. `$ mkimage -f \ ${WORK_DIR}/<path to DTS`

	file>/ventana-edk2-thunderhill.dts `ventana-thunderhill-edk2-fit.itb` **For information about FIT images and creating DTS files see FIT images.**

Table 11 : EDK II firmware build commands

5.6. QEMU image

Table 12 below lists the QEMU build input components and their paths. If one or more of these components are modified, the QEMU image needs to be rebuilt. Table 13 lists the commands for building QEMU.

QEMU build input components	Download command
Required packages for QEMU	`$ sudo apt install slirp libslirp-dev libvde-dev libvdeplug-dev libvte-2.91-dev libxen-dev liblzo2-dev ninja-build iasl`
QEMU source	`$ git clone -b release-v0.18.1 \ git@github.com:ventanamicro-dev/qemu.git`

Table 12 : QEMU build components

Build objective	Build commands

Build QEMU	`$ cd qemu`
	`$ ./configure --target-list=riscv64-softmmu`
	`$ make`

Build BMC QEMU	`$ cd qemu`
	`$ ./configure --target-list=arm-softmmu`
	`$ make`

Table 13 : QEMU build commands

5.7. FIT images

FIT (Flattened uImage Tree) image format (`.itb`) is used which combines multiple binaries (next stage bootloader, firmware, device tree blob, and kernel) into a single image file. FIT image also contains the metadata for each binary including the load address.

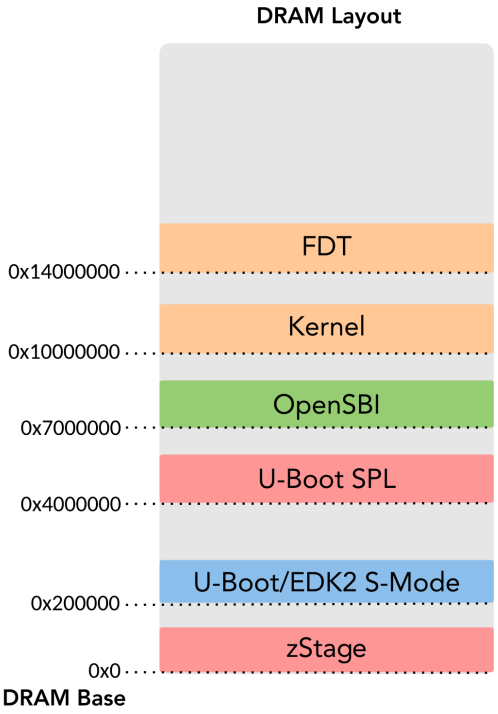
Ventana uses two FIT image variants depending on the bootloader, U-Boot bootloader in `u-boot.itb` and EDK II bootloader in `edk2.itb`.

The early stage bootloader reads the FIT image metadata and loads the different contained binaries to their respective load addresses as depicted in the DRAM layout image, below.

5.7.1. DRAM load addresses

For each FIT image component, the following are the ***offsets*** from the ***DRAM Base*:**

- zStage @ DRAM offset 0x0
- U-Boot/EDK II @ DRAM offset 0x200000
- U-Boot SPL @ DRAM offset 0x4000000
- OpenSBI @ DRAM offset 0x7000000
- Linux kernel @ DRAM offset 0x10000000
- Device Tree Binary (FDT) @ DRAM offset 0x14000000



5.7.2. FIT image layout for Ventana Synth V2

FIT image layout is a standard format which is detailed in U-Boot documentation - https://docs.u-boot.org/en/latest/usage/fit/source_file_format.html.

FIT image layout example for EDK II:

```
/dts-v1/;
/ {
    description = "Configuration file to create FIT image containing OpenSBI and
EDK2";
    #address-cells = <0x01>;
    fit,fdt-list = "of-list";

    images {
      edk2 {
        description = "EDK2";
        type = "standalone";
        os = "u-boot";
```

```
        arch = "riscv";
        compression = "none";
        load = <0x80200000>;
        entry = <0x80200000>;
        data = /incbin/("VENTANASYNTH.fd");
      };

      opensbi {
        description = "OpenSBI fw_dynamic Firmware";
        type = "firmware";
        os = "opensbi";
        arch = "riscv";
        compression = "none";
        load = <0x87000000>;
        entry = <0x87000000>;
        data = /incbin/("fw_dynamic.bin");
      };
    };

    configurations {
      default = "conf-1";
      conf-1 {
        description = "EDK2 and OpenSBI FIT";
        firmware = "opensbi";
        loadables = "edk2";
      };
    };
};
```

5.7.3. FIT image layout for Ventana Thunderhill V2

FIT image layout is a standard format which is detailed in U-Boot documentation - https://docs.u-boot.org/en/latest/usage/fit/source_file_format.html.

FIT image layout example for EDK II:

```
/dts-v1/;
/ {
```

```
        description = "Configuration file to create FIT image containing OpenSBI
and EDK2";
        #address-cells = <0x02>;
        fit,fdt-list = "of-list";
        images {
                edk2 {
                        description = "EDK2";
                        type = "standalone";
                        os = "U-Boot";
                        arch = "riscv";
                        compression = "none";
                        load = <0x00000004 0x00200000>;
                        entry = <0x00000004 0x00200000>;
                        data = /incbin/("THUNDERHILL.fd"));
                };
                opensbi {
                        description = "OpenSBI fw_dynamic Firmware";
                        type = "firmware";
                        os = "opensbi";
                        arch = "riscv";
                        compression = "none";
                        load = <0x00000004 0x07000000>;
                        entry = <0x00000004 0x07000000>;
                        data = /incbin/("fw_dynamic.bin");
                };
        };
        configurations {
                default = "conf-1";
                conf-1 {
                        description = "EDK2 and OpenSBI FIT";
                        firmware = "opensbi";
                        loadables = "edk2";
                };
        };
};
```

5.8. BIOS SD Card Image

The following sub-sections explain how to create minimal BIOS SD Card Image which is just a few MBs and only contains a ESP partition with fw.img (FIT) and zstage.bin for various Ventana platforms.

5.8.1. U-boot BIOS SD Card Image

1. Run following commands to clone genimage and compile from opensource

```
$ git clone https://github.com/pengutronix/genimage.git
$ cd genimage
$ ./autogen.sh
$ ./configure CFLAGS='-g -O0' --prefix=/usr
$ make
$ cd ..
```

2. Run following commands to clone genfatfs and compile from opensource

```
$ git clone https://github.com/NodeOS/genfatfs.git
$ cd genfatfs
$ make
$ cd ..
```

3. Create FIRMWARE directories and copy the FIT images and zStage binaries

```
$ rm -rf sdcard
$ mkdir -p sdcard/bios_esp/FIRMWARE/ventana/thunderhill-vx
$ mkdir -p sdcard/bios_esp/FIRMWARE/ventana/thunderhill-v2
$ mkdir -p sdcard/bios_esp/FIRMWARE/ventana/synth-vx

$ cp
$WORK_DIR/ventana-sw/platform/ventana-thunderhill-vx/u-boot/ventana-thunderhill-u
-boot-fit-${VENTANA_SW_VERSION}.itb
sdcard/bios_esp/FIRMWARE/ventana/thunderhill-vx/fw.itb

$ cp $WORK_DIR/ventana-sw/platform/ventana-thunderhill-v2/zstage/zstage.bin
sdcard/bios_esp/FIRMWARE/ventana/thunderhill-v2/zstage.bin
```

```
$ cp
$WORK_DIR/ventana-sw/platform/ventana-synth-vx/u-boot/ventana-synth-u-boot-fit-${
VENTANA_SW_VERSION}.itb sdcard/bios_esp/FIRMWARE/ventana/synth-vx/fw.itb

$ dd if=/dev/zero of=./sdcard/bios_esp.img bs=1024 count=114688

$ ./genfatfs/genfatfs -d ./sdcard/bios_esp -b 114688 ./sdcard/bios_esp.img
```

4. Create a config file for BIOS SD card image

```
$ printf "image bios_riscv64_ventana.img {\n\
        size = 128M\n\
\n\
        hdimage {\n\
                partition-table-type = gpt\n\
        }\n\
\n\
        partition esp {\n\
                image = \"bios_esp.img\"\n\
                offset = 8M\n\
                size = 112M\n\
                partition-type-uuid = C12A7328-F81F-11D2-BA4B-00A0C93EC93B\n\
        }\n\
}\n" > ./sdcard/bios_riscv64_ventana.cfg
```

5. Create the BIOS SD card image inside the sdcard directory

```
$ ./genimage/genimage --config ./sdcard/bios_riscv64_ventana.cfg --inputpath
./sdcard --outputpath ./sdcard

$ ls sdcard/bios_u-boot_riscv64_ventana.img
```

5.8.2. EDK2 BIOS SD Card Image

1. Run following commands to clone genimage and compile from opensource

```
$ git clone https://github.com/pengutronix/genimage.git
$ cd genimage
$ ./autogen.sh
$ ./configure CFLAGS='-g -O0' --prefix=/usr
$ make
$ cd ..
```

2. Run following commands to clone genfatfs and compile from opensource

```
$ git clone https://github.com/NodeOS/genfatfs.git
$ cd genfatfs
$ make
$ cd ..
```

3. Create FIRMWARE directories and copy the FIT images and zStage binaries

```
$ rm -rf sdcard

$ mkdir -p sdcard/bios_esp/FIRMWARE/ventana/thunderhill-vx
$ mkdir -p sdcard/bios_esp/FIRMWARE/ventana/thunderhill-v2
$ mkdir -p sdcard/bios_esp/FIRMWARE/ventana/synth-vx

$ cp
$WORK_DIR/ventana-sw/platform/ventana-thunderhill-vx/edk2/ventana-thunderhill-edk
2-fit-${VENTANA_SW_VERSION}.itb
sdcard/bios_esp/FIRMWARE/ventana/thunderhill-vx/fw.itb

$ cp $WORK_DIR/ventana-sw/platform/ventana-thunderhill-v2/zstage/zstage.bin
sdcard/bios_esp/FIRMWARE/ventana/thunderhill-v2/zstage.bin

$ cp
$WORK_DIR/ventana-sw/platform/ventana-synth-vx/edk2/ventana-synth-edk2-fit-${VENT
ANA_SW_VERSION}.itb sdcard/bios_esp/FIRMWARE/ventana/synth-vx/fw.itb

$ dd if=/dev/zero of=./sdcard/bios_esp.img bs=
024 count=114688
$ ./genfatfs/genfatfs -d ./sdcard/bios_esp -b 114688 ./sdcard/bios_esp.img
```

4. Create a config file for BIOS SD card image

```
$ printf "image bios_riscv64_ventana.img {\n\
        size = 128M\n\
\n\
        hdimage {\n\
                partition-table-type = gpt\n\
        }\n\
\n\
        partition esp {\n\
                image = \"bios_esp.img\"\n\
                offset = 8M\n\
                size = 112M\n\
                partition-type-uuid = C12A7328-F81F-11D2-BA4B-00A0C93EC93B\n\
        }\n\
}\n" > ./sdcard/bios_riscv64_ventana.cfg
```

5. Create the BIOS SD card image inside the sdcard directory

```
$ ./genimage/genimage --config ./sdcard/bios_riscv64_ventana.cfg --inputpath
./sdcard --outputpath ./sdcard

$ ls sdcard/bios_edk2_riscv64_ventana.img
```

5.9. Cloud Init ISO Image

The `$WORK_DIR/ventana-sw/ventana-cloud-init.sh` available as part of pre-built binaries can be used to re-create ventana cloud-init ISO images. This script requires the `genisoimage` package so make sure it is installed before running `ventana-cloud-init.sh`.

To create `$WORK_DIR/ventana-sw/ventana-cloud-ubuntu.iso`, use the following commands:

```
$  cd $WORK_DIR/ventana-sw
$  ./ventana-cloud-init.sh \
   -n ventana-cloud-ubuntu.iso \
   -a “build-essential python3-venv pkg-config libglib2.0-dev
   python3-sphinx-rtd-theme python3-sphinx ninja-build pkg-config libglib2.0-dev” \
   -p /opt/ventana/ventana-apps/bin \
   -f http://10.0.2.2:8000/linux-image-6.15.0-ventana-0.18.1_riscv64.deb@debinst \
   -f http://10.0.2.2:8000/linux-source-6.15.0-ventana-0.18.1_riscv64.deb@debinst \
   -f http://10.0.2.2:8000/ventana-apps.tar.xz@unxz \
   -f http://10.0.2.2:8000/linux-headers-6.15.0-ventana-0.18.1_riscv64.deb@debinst
```

To create `$WORK_DIR/ventana-sw/ventana-cloud-ubuntu-toolchain.iso`, use the following commands:

```
$  cd $WORK_DIR/ventana-sw
$  ./ventana-cloud-init.sh \
   -n ventana-cloud-ubuntu-toolchain.iso \
   -a “build-essential python3-venv pkg-config libglib2.0-dev
   python3-sphinx-rtd-theme python3-sphinx ninja-build pkg-config libglib2.0-dev” \
   -p /opt/ventana/ventana-native-toolchain-2025.05.22/bin:/opt/perf/bin \
   -p /opt/ventana/ventana-apps/bin \
   -f http://10.0.2.2:8000/linux-image-6.15.0-ventana-0.18.1_riscv64.deb@debinst \
   -f http://10.0.2.2:8000/linux-source-6.15.0-ventana-0.18.1_riscv64.deb@debinst \
   -f http://10.0.2.2:8000/ventana-apps.tar.xz@unxz \
   -f http://10.0.2.2:8000/linux-headers-6.15.0-ventana-0.18.1_riscv64.deb@debinst \
   -f http://10.0.2.2:8000/ventana-native-toolchain-2025.05.22.tar.xz@unxz
```

6. Virtualization using KVM

It is possible to launch a KVM guest using QEMU-KVM and QEMU-KVM with libvirt. We will create a lightweight kernel and disk image for it, then we will go through the required setup for the emulated RISC-V host.

6.1. Prepare RISC-V host for Linux KVM

Our RISC-V host will be the same QEMU-emulated Synth instance we launched in Run images on QEMU.

After booting it we'll need to make some changes. All these steps are made inside this emulated RISC-V host.

6.1.1. Setup KVM module

When using the preinstalled server image as in 3. Booting and Running, this step has already been done for you, and you can proceed to the next section.

Load the KVM module and change its permission to make it usable by libvirt later on:

```
$  sudo modprobe kvm
$  sudo chmod o+rw /dev/kvm
```

To make the KVM module always load during boot, add a "kvm" line at the end of the /etc/modules-load.d/modules.conf file:

```
# /etc/modules: kernel modules to load at boot time.
#
# This file contains the names of kernel modules that should be loaded
# at boot time, one per line. Lines beginning with "#" are ignored.
# Parameters can be specified after the module name.
kvm
```

6.1.2. Install and configure libvirt

When using the preinstalled server image as in 3. Booting and Running, this step has already been done for you, and you can proceed to the next section.

We will use the distro libvirt to launch a RISC-V KVM guest.Install libvirt:

```
$  sudo apt install libvirt-daemon-system
```

6.2. Guest kernel and disk image

When using the preinstalled server image as in 3. Booting and Running, this step has already been done for you, and you can proceed to the next section.

We need a kernel and a disk image to boot a QEMU-KVM guest.

- For the kernel image we'll use the same kernel binary provided in the emulated RISC-V host under the /boot directory. Save it to a new 'images' directory inside the home dir of the emulated RISC-V host:

```
$  mkdir images
$  cp /boot/vmlinuz-6.15.0-ventana images/Image
```

- For the disk image, follow the instructions on host machine https://github.com/kvm-riscv/howto/wiki/KVM-RISCV64-on-QEMU#6-build-host-rootfs-containing-kvm-risc-v-module-kvmtool-and-guest-linux to build a guest image. At the end of the process described in that link this command will generate an 'ext2' rootfs disk.

```
$  mke2fs -d busybox-1.33.1-kvm-riscv64/_install ./rootfs.ext2 32M
```

- Copy the generated `rootfs.ext2` file to the same `images` folder inside the emulated RISC-V host where we copied the kernel image.

6.3. Launch KVM Guest using QEMU

Assuming the guest kernel and disk image are in the `/home/ventana/images` dir, and the QEMU binary built with Ventana source code at `/home/ventana/qemu`, the following command line will boot a QEMU guest:

```
$  sudo /usr/bin/qemu-system-riscv64 -machine virt -accel kvm -m 1G -smp 2
   -serial stdio -nodefaults -nographic -kernel /home/ventana/images/Image
   -append "rootwait root=/dev/vda ro" -drive
   file=/home/ventana/images/rootfs.ext2,format=raw,if=virtio
```

The buildroot image credentials are 'root' and blank password. To shutdown the guest run the following command from within it:

```
#  poweroff -f
```

6.4. Launch KVM Guest using libvirt

When using the preinstalled server image as in 3. Booting and Running, this step has already been done for you, and you can proceed to the next section.

We'll create a libvirt QEMU-KVM domain that mirrors the same guest we launched in the previous section using the QEMU command line.

- Copy kernel image and rootfs to `/var/lib/libvirt/images/` directory and qemu to `/usr/bin/`:

```
$  sudo cp /boot/vmlinuz-6.15.0-ventana
   /var/lib/libvirt/images/Image

$  sudo cp images/rootfs.ext2 /var/lib/libvirt/images

$  sudo cp /home/ventana/qemu/build/qemu-system-riscv64 /usr/bin/
```

- Create an XML file that describes the domain. In this example we called the file 'vm.xml'. The domain being created is called 'qemu-kvm-vm':

```
$  cat vm.xml
<domain type='kvm'>
  <name>qemu-kvm-vm</name>
  <memory unit='G'>1</memory>
  <vcpu placement='static'>2</vcpu>
  <os>
    <type arch='riscv64' machine='virt'>hvm</type>
    <kernel>/var/lib/libvirt/images/Image</kernel>
    <cmdline>rootwait root=/dev/vda ro</cmdline>
    <boot dev='hd'/>
  </os>
  <devices>
    <emulator>/usr/bin/qemu-system-riscv64</emulator>
    <disk type='file' device='disk'>
      <driver name='qemu' type='raw'/>
      <source file='/var/lib/libvirt/images/rootfs.ext2'/>
```

```
      <target dev='vda' bus='virtio'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x01'
function='0x0'/>
    </disk>
    <controller type='pci' index='0' model='pcie-root'/>
    <interface type='network'>
      <source network='default'/>
      <model type='virtio'/>
    </interface>
    <serial type='pty'>
      <target type='system-serial' port='0'>
        <model name='16550a'/>
      </target>
    </serial>
    <console type='pty'>
      <target type='serial' port='0'/>
    </console>
    <memballoon model='none'/>
  </devices>
</domain>
```

- Use the XML to create the domain inside libvirt. We do that using libvirt command line tool 'virsh':

```
$ virsh define vm.xml
Domain 'qemu-kvm-vm' defined from vm.xml
```

- Start the network

```
$ virsh net-autostart default
```

- Start the domain:

```
$ virsh start qemu-kvm-vm
Domain 'qemu-kvm-vm' started
```

- The domain status can be checked using 'virsh list':

```
$  virsh list --all
```

```
Id   Name           State
------------------------------
1    qemu-kvm-vm    running
```

- The virtual machine console is accessed via the 'virsh console' command:

```
$  virsh console qemu-kvm-vm
Connected to domain 'qemu-kvm-vm'
Escape character is ^] (Ctrl + ])

Welcome to Buildroot
buildroot login: root
# cat /proc/cpuinfo
processor   : 0
hart        : 0
isa         : rv64imafdc_sstc_svinval_svpbmt_zicbom_zihintpause
mmu         : sv48
mvendorid   : 0x61f
marchid           : 0x8000000000010000
mimpid            : 0x111
(...)
```

Note that the console can be released using the CTRL +] keyboard shortcut.

- To shutdown the guest we can either issue a 'poweroff' command or a libvirt command. We'll shutdown it via 'virsh destroy':

```
$ virsh destroy qemu-kvm-vm
Domain 'qemu-kvm-vm' destroyed
```

libvirt has a rich set of commands for guest management and operation. See https://wiki.libvirt.org/UbuntuKVMWalkthrough.html for more information.

7. Performance counters

The Ventana Veyron PMU supports various events and metrics that can be counted using the perf tool in Linux.

Use the "perf list" command to display the list of supported events (Hardware, cache, and software) and metrics.

```
$  perf list
```

Following is the complete list of named events:

```
  branch-instructions OR branches                    [Hardware event]
  branch-misses                                      [Hardware event]
  cache-misses                                       [Hardware event]
  cache-references                                   [Hardware event]
  cpu-cycles OR cycles                               [Hardware event]
  instructions                                       [Hardware event]
  stalled-cycles-backend OR idle-cycles-backend      [Hardware event]
  stalled-cycles-frontend OR idle-cycles-frontend    [Hardware event]
  alignment-faults                                   [Software event]
  bpf-output                                         [Software event]
  cgroup-switches                                    [Software event]
  context-switches OR cs                             [Software event]
  cpu-clock                                          [Software event]
  cpu-migrations OR migrations                       [Software event]
  dummy                                              [Software event]
  emulation-faults                                   [Software event]
  major-faults                                       [Software event]
  minor-faults                                       [Software event]
  page-faults OR faults                              [Software event]
  task-clock                                         [Software event]

  L1-dcache-loads OR cpu/L1-dcache-loads/
  L1-dcache-load-misses OR cpu/L1-dcache-load-misses/
  L1-dcache-stores OR cpu/L1-dcache-stores/
  L1-dcache-store-misses OR cpu/L1-dcache-store-misses/
  L1-dcache-prefetch-misses OR cpu/L1-dcache-prefetch-misses/
```

```
  L1-icache-loads OR cpu/L1-icache-loads/
  L1-icache-load-misses OR cpu/L1-icache-load-misses/
  L1-icache-prefetches OR cpu/L1-icache-prefetches/
  L1-icache-prefetch-misses OR cpu/L1-icache-prefetch-misses/
  LLC-loads OR cpu/LLC-loads/
  LLC-load-misses OR cpu/LLC-load-misses/
  LLC-stores OR cpu/LLC-stores/
  LLC-store-misses OR cpu/LLC-store-misses/
  LLC-prefetches OR cpu/LLC-prefetches/
  LLC-prefetch-misses OR cpu/LLC-prefetch-misses/
  dTLB-loads OR cpu/dTLB-loads/
  dTLB-load-misses OR cpu/dTLB-load-misses/
  dTLB-stores OR cpu/dTLB-stores/
  dTLB-store-misses OR cpu/dTLB-store-misses/
  dTLB-prefetches OR cpu/dTLB-prefetches/
  dTLB-prefetch-misses OR cpu/dTLB-prefetch-misses/
  iTLB-loads OR cpu/iTLB-loads/
  iTLB-load-misses OR cpu/iTLB-load-misses/
  branch-loads OR cpu/branch-loads/
  branch-load-misses OR cpu/branch-load-misses/
```

firmware:

```
fw_access_load
       [Load access trap event]
 fw_access_store
       [Store access trap event]
 fw_fence_i_received
       [Received FENCE.I request from other HART event]
 fw_fence_i_sent
       [Sent FENCE.I request to other HART event]
 fw_hfence_gvma_received
       [Received HFENCE.GVMA request from other HART event]
 fw_hfence_gvma_sent
       [Sent HFENCE.GVMA request to other HART event]
 fw_hfence_gvma_vmid_received
       [Received HFENCE.GVMA with VMID request from other HART event]
```

```
 fw_hfence_gvma_vmid_sent
       [Sent HFENCE.GVMA with VMID request to other HART event]
 fw_hfence_vvma_asid_received
       [Received HFENCE.VVMA with ASID request from other HART event]
 fw_hfence_vvma_asid_sent
       [Sent HFENCE.VVMA with ASID request to other HART event]
 fw_hfence_vvma_received
       [Received HFENCE.VVMA request from other HART event]
 fw_hfence_vvma_sent
       [Sent HFENCE.VVMA request to other HART event]
 fw_illegal_insn
       [Illegal instruction trap event]
 fw_ipi_received
       [Received IPI from other HART event]
 fw_ipi_sent
       [Sent IPI to other HART event]
 fw_misaligned_load
       [Misaligned load trap event]
 fw_misaligned_store
       [Misaligned store trap event]
 fw_set_timer
       [Set timer event]
 fw_sfence_vma_asid_received
       [Received SFENCE.VMA with ASID request from other HART event]
 fw_sfence_vma_received
       [Sent SFENCE.VMA with ASID request to other HART event]
 fw_sfence_vma_sent
       [Sent SFENCE.VMA request to other HART event]
```

hwmon:

```
temp_composite OR temp1
       [Temperature in unit nvme named Composite.
alarm=0'C,crit=99.85'C,max=69.85'C,min=-273.15'C. Unit: hwmon_nvme]
```

microarch:

```
  arch_abort
```

```
       [Architectural trap including those done speculatively. Unit: cpu]
  btb_hit_event1
       [Branch target buffer (BTB) hit of fetch blocks by the prediction
unit. Must to added to BTB_HIT_EVENT2 for total BTB
        hits. Unit: cpu]
  btb_hit_event2
       [Branch target buffer (BTB) hit of fetch blocks by the prediction
unit. Must be added to BTB_HIT_EVENT1 for total BTB
        hits. Unit: cpu]
  cache_l1d_read_access
       [DL1 load access. Unit: cpu]
  csr_access
       [CSR op issued to the shared unit. Unit: cpu]
  cyc_no_disp_empty
       [Dispatch empty. Unit: cpu]
  disp_count1_ops
       [Mop dispatched. Unit: cpu]
  dl2_demand_read
       [DL2 read access. Unit: cpu]
  dl2_demand_read_miss
       [DL2 read miss. Unit: cpu]
  dl2_ext_snoop
       [Level 2 memory unit (L2M) inbound snoop. Unit: cpu]
  dl2_orb_update_request
       [DL2 outstanding read buffer update request. Unit: cpu]
  dl2_prefetch_l1
       [DL2 access due to L1 prefetch. Unit: cpu]
  dl2_prefetch_l1_miss
       [DL2 miss due to L1 prefetch. Unit: cpu]
  dl2_prefetch_l2_miss
       [DL2 miss due to L2 prefetch. Unit: cpu]
  dl2_write
       [DL2 write access. Unit: cpu]
  dl2_write_miss
       [DL2 write miss. Unit: cpu]
  fe_arch_abort
       [Microarchitectural abort (including those done speculatively).
Unit: cpu]
  fe_mp_btb_f_abort
```

```
       [Abort due to shortening of max-length sequential fetch blocks.
Unit: cpu]
  fe_mp_btb_nf_abort
       [Other fetch block related decoder uarch abort (as a single metric).
Unit: cpu]
  fe_mp_ebr_t_abort
       [Abort due to embedded conditional branch taken. Unit: cpu]
  full_sync_abort
       [Abort due to full synchronization event. Unit: cpu]
  il2_ext_snoop
       [Instruction fetch unit inbound snoop. Unit: cpu]
  l2_prefetch_dl2
       [DL2 access due to L2 prefetch. Unit: cpu]
  l2m_hfence
       [hfence.* instruction retired. Unit: cpu]
  l2m_sfence
       [sfence.* instruction retired. Unit: cpu]
  load_fwding_viol_abort
       [Abort due to store->load forwarding violation. Unit: cpu]
  lsu_fwd_prog_act
       [DL1_MISS - LSU forward progress activation. Unit: cpu]
  lsu_memop_stcmt
       [Store memory operation commit. Unit: cpu]
  lsu_replay
       [Memory operation replay due to miss,lack of resources,store
unforwardable scenario,etc. Unit: cpu]
  lsu_replay_noid
       [DL1 miss - LSU replay without a valid memdep/replay ID. Unit: cpu]
  lsu_retry
       [LSU memory operation (load,store,atomic memory operation,or fence)
retry. Unit: cpu]
  pap_reuse_abort
       [Abort due to physical address proxy reuse. Unit: cpu]
  pred_fblk_termtype_event2
       [Branch target buffer (BTB) misprediction of fetch blocks by the
prediction unit. Unit: cpu]
  retire_type_all
       [Mop retired. Unit: cpu]
  retired_branch_nonret_ibr
```

```
       [Indirect branch (with/without misprediction). Unit: cpu]
  retired_cbr
       [Conditional branch (with/without misprediction) retired. Unit: cpu]
  retired_cbr_mispred
       [Conditional branch with misprediction retired. Unit: cpu]
  retired_dbr_mispred
       [Mispredicted direct branch (J/JAL) retired. Unit: cpu]
  retired_nonret_ibr_mispred
       [Mispredicted indirect branch (JR/JALR) retired. Unit: cpu]
  retired_ret
       [Indirect branch (with/without misprediction) retired. Unit: cpu]
  retired_ret_mispred
       [Indirect branch with misprediction retired. Unit: cpu]
```

tool:

```
  duration_time
       [Wall clock interval time in nanoseconds. Unit: tool]
  has_pmem
       [1 if persistent memory installed otherwise 0. Unit: tool]
  num_cores
       [Number of cores. A core consists of 1 or more thread,with each
thread being associated with a logical Linux CPU. Unit:
        tool]
  num_cpus
       [Number of logical Linux CPUs. There may be multiple such CPUs on a
core. Unit: tool]
  num_cpus_online
       [Number of online logical Linux CPUs. There may be multiple such
CPUs on a core. Unit: tool]
  num_dies
       [Number of dies. Each die has 1 or more cores. Unit: tool]
  num_packages
       [Number of packages. Each package has 1 or more die. Unit: tool]
  smt_on
       [1 if simultaneous multithreading (aka hyperthreading) is enable
otherwise 0. Unit: tool]
  system_time
```

```
       [System/kernel time in nanoseconds. Unit: tool]
  user_time
       [User (non-kernel) time in nanoseconds. Unit: tool]
  rNNN                                                    [Raw event descriptor]

cpu/event=0..0xffffffffffff,firmware=0..3,config=0..0xffffffffffffffff,.../
modifier[Raw event descriptor]
       [(see 'man perf-list' or 'man perf-record' on how to encode it)]
  breakpoint//modifier                                    [Raw event descriptor]
  hwmon_nvme//modifier                                    [Raw event descriptor]
  kprobe/retprobe/modifier                                [Raw event descriptor]
  software//modifier                                      [Raw event descriptor]
  tool//modifier                                          [Raw event descriptor]
  tracepoint//modifier                                    [Raw event descriptor]
  uprobe/ref_ctr_offset=0..0xffffffff,retprobe/modifier[Raw event
descriptor]
  mem:<addr>[/len][:access]                               [Hardware breakpoint]
```

metrics:

```
TopDown:
  DispatchSlots
       [Dispatch slots]
  DispatchStallSlots
       [Dispatch stall slots]
  DispatchUtilization
       [Dispatch utilization]
  DispatchZeroRate
       [Dispatch zero ops rate]
  OpAbortRate
       [Op abort rate]
  OpsAborted
       [Ops aborted]
```

7.1. Counting with perf stat

For any of the supported events, perf can keep a running count during process execution. In counting modes, the occurrences of events are simply aggregated and presented on the

standard output at the end of an application run. The list of events to be counted can be provided to perf as named events, raw events, or a mix of both.

7.2. Named events

```
$  perf stat -e cycles -e fw_misaligned_store -e pap_reuse_aborts <command>
```

where `<command>` is the application that needs to be profiled and the names specified after `-e` are event names listed in `perf list` command. The event list can contain a mix of standard hardware events, firmware events and microarch events.

7.3. Raw events

Other than the commonly used named events, the Ventana V2 PMUs also support a large set of raw events which can be counted using the `perf` tool. For further information about such events, please contact support@ventanamicro.com.

8. Secure Boot

The SDK releases (0.18.1 and onwards) are being shipped with all necessary components having the desired changes/ configuration to enable the end to end **Secure Boot** feature starting from the zStage boot till the EDK2 gets booted.

Note:- This feature is supported ONLY on **Ventana Synth** platforms (all variants).

8.1. Generate Asymmetric Cryptographic Keys

An important prerequisite to be able to achieve Secure Boot is to digitally sign the relevant images/ data and to do that it is MUST to have a pair of asymmetric cryptographic keys i.e., a private-public key pair. To generate such key pairs, the normal practice is to use **OpenSSL** which is supported on multiple platforms and Operating Systems including Linux, Microsoft Windows, MacOS.

The images to be signed are OpenSBI + EDK2 (packed together in FIT format) and zStage (standalone image + metadata). For FIT Image signing, only RSA based keys are to be used while for zStage Image signing, only ECC (curve secp384r1) based keys are to be used.

RSA asymmetric cryptographic key generation example using **`openssl`**:

```
$  export SBOOT_DIR=$WORK_DIR/ventana-sw/platform/ventana-synth-vx/secure-boot
$  cd $SBOOT_DIR/keys

$  openssl genrsa -F4 -out ventana_rsa-0.key 3072
$  openssl req -batch -new -x509 -key ventana_rsa-0.key -out ventana_rsa-0.crt
```

ECC asymmetric cryptographic key generation example using **`openssl`**:

```
$  openssl ecparam -name secp384r1 -genkey -noout -out ventana_ecc-prvkey-0.pem
$  openssl ec -pubout -in ventana_ecc-prvkey-0.pem -out ventana_ecc-pubkey-0.pem
```

For the purpose of matching with illustrations in this manual, it is hereby assumed that 3 sets of RSA keys i.e., `ventana_rsa-0/1/2.key` and their corresponding `ventana_rsa-0/1/2.crt` and 4 sets of ECC keys `ventana_ecc-prvkey-0/1/2/3.pem` and their corresponding `ventana_ecc-pubkey-0/1/2/3.pem` have been created using the commands written above.

8.2. Create Signed FIT Image

To generate a signed FIT image, the image tree source file needs to be created by including appropriate **`hash`** and **`signature`** nodes.

8.2.1. FIT image layout extension for Ventana Synth V2 and E2

Add **`hash`** nodes at **`images`** level and **`signature`** nodes at **`images`** as well as **`configurations`** level in the FIT image layout source file with appropriate parameter values.

Signed FIT image layout example for EDK II:

```
/*
 * FIT Configuration source file to create signed
 * FIT image with OpenSBI and EDK2 UEFI binaries.
```

```
 */
/dts-v1/;
/ {
	description = "Configuration to load OpenSBI before EDK2";
	#address-cells = <2>;

	images {
		edk2 {
			description = "EDK2";
			type = "standalone";
			os = "U-Boot";
			arch = "riscv";
			compression = "none";
			load = <0x80200000>;
			entry = <0x80200000>;
			data = /incbin/("./uefi.bin");

			hash-2 {
				algo = "sha512";
			};
			signature-2 {
				algo = "sha384,rsa3072";
				key-name-hint = "ventana_rsa-2";
			};
		};

		opensbi {
			description = "OpenSBI Dynamic Firmware";
			type = "firmware";
			os = "opensbi";
			arch = "riscv";
			compression = "none";
			load = <0x87000000>;
			entry = <0x87000000>;
			data = /incbin/("./../../../fw_dynamic.bin");

			hash-1 {
				algo = "sha512";
			};
			signature-1 {
				algo = "sha384,rsa3072";
				key-name-hint = "ventana_rsa-1";
			};
```

```
                };
        };

        configurations {
                default = "conf-0";
                conf-0 {
                        description = "FIT Image and Signature Configuration";
                        firmware = "opensbi";
                        loadables = "edk2";

                        /*
                         * Add at least one signature at configurations level
                         * including both opensbi and edk2 under sign-images.
                         */
                        signature-0 {
                                algo = "sha384,rsa3072";
                                key-name-hint = "ventana_rsa-0";
                                sign-images = "firmware","loadables";
                        };
                };
        };
};
```

- **hash** nodes under **images/edk2** and **images/opensbi** are mandatory to be added when **sign-images** (under **signature-0** node in **configurations**) include both.
- **key-name-hint** should be from the name of the RSA keys already generated. Please ensure to generate all keys used as described in Sec 8.1.
- **signature** nodes in **images/edk2** and **images/opensbi** are required to be either included in both or excluded from both.

Let the name of the above FIT image tree layout source sample file be **signfit-edk2.its** saved inside **$WORK_DIR/ventana-sw/platform/ventana-synth-vx/edk2**. The paths of the OpenSBI and EDK2 images as specified by the **data** field in the configuration file above are relative to the **$WORK_DIR/ventana-sw/platform/ventana-synth-vx/edk2** folder.

Note:- Better to keep the keys used for signing at **configurations** level and that used for signing at **images** level different. The keys used for signing at **images** level can be different or the same. When signing at **images** level, then please ensure that all images are signed. Multiple **signature** nodes having **different** keys may be added at **configurations** level with single image or multiple images included under **sign-images**.

8.2.2. Generate unsigned FIT image

Let the name of the FIT image output file be **`SFitImage.`**

Unsigned FIT image creation example using **`mkimage`**:

```
$  cd $WORK_DIR/ventana-sw/platform/ventana-synth-vx
$  mkimage -D "-I dts -O dtb -p 2000" -f \
   $WORK_DIR/ventana-sw/platform/ventana-synth-vx/edk2/signfit-edk2.its \
   $WORK_DIR/ventana-sw/platform/ventana-synth-vx/edk2/SFitImage
```

Sample output of the last **mkimage** command:

```
FIT description: Configuration to load OpenSBI before U-Boot
Created:         Wed Mar  5 09:38:24 2025
 Image 0 (edk2)
  Description:  EDK2
  Created:      Wed Mar  5 09:38:24 2025
  Type:         Standalone Program
  Compression:  uncompressed
  Data Size:    8388608 Bytes = 8192.00 KiB = 8.00 MiB
  Architecture: RISC-V
  Load Address: 0x80200000
  Entry Point:  unavailable
  Hash algo:    sha512
  Hash value:   b88f5824e1c7924a8ecdda8f377ca25916d0286ad147177827389e6300ae65dae1e063a65eb1f2b1af4c4e051
0a6cba567d151d60301e692f13af9520a0ae680
  Sign algo:    sha384,rsa3072:ventana_rsa-2
  Sign value:   unavailable
  Timestamp:    unavailable
 Image 1 (opensbi)
  Description:  OpenSBI Dynamic Firmware
  Created:      Wed Mar  5 09:38:24 2025
  Type:         Firmware
  Compression:  uncompressed
  Data Size:    274920 Bytes = 268.48 KiB = 0.26 MiB
  Architecture: RISC-V
  OS:           RISC-V OpenSBI
  Load Address: 0x87000000
  Hash algo:    sha512
  Hash value:   67ade658ed95f0a6b731594c16bb853cdff431f8a739e67885b38967961bb312b3fbccfc4ec6c975628696c8b
ef5dd36a342c0bcf22f7330a6d8ecefce2b1a38
  Sign algo:    sha384,rsa3072:ventana_rsa-1
  Sign value:   unavailable
  Timestamp:    unavailable
 Configuration 0 (conf-0)
  Description:  FIT Image and Signature Configuration
  Kernel:       unavailable
  Firmware:     opensbi
  Loadables:    edk2
  Sign algo:    sha384,rsa3072:ventana_rsa-0
  Sign value:   unavailable
  Timestamp:    unavailable
```

Note : Screenshot is illustrative

As can be seen in the sample output, corresponding to all the **hash** nodes added in the **signfit-edk2.its** file, the **Hash algo** as well as the **Hash value** fields has been filled with values whereas corresponding to all the **signature** nodes, only the **Sign algo** field has been filled with algorithm name and the key name to be used for signing, while the **Sign value** field shows **unavailable** which means that the FIT image is currently unsigned.

8.2.3. Generate FIT Public Key DTB file

Create a sample public key device tree source (.dts) file and use it to generate a device tree blob (.dtb) file to be used for public keys storage in DTB format.

```
/*
 * DTS source file with blank content initially to generate a DTB file
 * that will hold public keys information
 */
/dts-v1/;
/ {
      description = "Acts as storage for public key(s) information";
};
```

Let the name of the above sample public key dts file be **ventana_rsa-pubkeys.dts** and the name of the corresponding public key dtb file (to be generated) be **ventana_rsa-pubkeys.dtb**, both of which are saved under the **$SBOOT_DIR/keys** folder.

Blank DTB file generation example using **dtc**:

```
$  dtc -I dts -O dtb -o \
   $SBOOT_DIR/keys/ventana_rsa-pubkeys.dtb \
   $SBOOT_DIR/keys/ventana_rsa-pubkeys.dts
```

8.2.4. Signing the FIT image

Sign the FITImage using **mkimage** by providing the path to the private keys (-k option) and the dtb file to store the corresponding public keys (-K option).

Signed FIT image creation example using **mkimage**:

```
$  mkimage -D "-I dts -O dtb -p 2000" -F \
   -k $SBOOT_DIR/keys \
   -K $SBOOT_DIR/keys/ventana_rsa-pubkeys.dtb \
   -r $WORK_DIR/ventana-sw/platform/ventana-synth-vx/edk2/SFitImage
```

Sample output of the last **mkimage** command:

```
FIT description: Configuration to load OpenSBI before U-Boot
Created:         Wed Mar  5 09:38:24 2025
 Image 0 (edk2)
  Description:  EDK2
  Created:      Wed Mar  5 09:38:24 2025
  Type:         Standalone Program
  Compression:  uncompressed
  Data Size:    8388608 Bytes = 8192.00 KiB = 8.00 MiB
  Architecture: RISC-V
  Load Address: 0x80200000
  Entry Point:  unavailable
  Hash algo:    sha512
  Hash value:   b88f5824e1c7924a8ecdda8f377ca25916d0286ad147177827389e6300ae65dae1e063a65eb1f2b1af4c4e051
0a6cba567d151d60301e692f13af9520a0ae680
  Sign algo:    sha384,rsa3072:ventana_rsa-2
  Sign value:   09b6c2173785ca6f12c59ae3d598f52c6507a7c9f89dd4364f25b922121603ecd9686b3d8cce55129677298c5
ecdac229513aa354171eefcea75a794fde65d80893b3217981e12b3503eea4fde3d3ebafb0cd7d49e5fc1337127b12b29bbc8e1f9
8a7e87c942bcb64f1102f9e0ae10681a92317b72a585f1b427764a01dc8e7594ad0e8563c67f2f2e653bd3b3d6e806781f8480b60
9c0f2f6d2ef4564ba71a1d623e4326582562a9cc0e6216d23acdf26920f16ffcaa94b4d36687f210b97e9c10fdbdfd3a8a6615354
e5ee04659942a4a7f290a8e2af4a2f4733493418424fde4e4e5474a1234663688d4055f7a5c8382bad95abf0ae8fa8f37221b4294
605444ac7ae25bc5051f7a9b07b0145ae884cb1800291d1e5580133d8d5f51b2fb96708f85d3dbcd05620795bb591c28c3e02683c
f358bdfb3c59fd8cff05394700be8f1f4401be3343cc39ad48807c16de6d30268cb45ff828c7c34c8fbb0408ad9706570fde627c2
a021caa2e314331467e266c84eb91b992a6724b9912855297
  Timestamp:    Wed Mar  5 09:55:40 2025
 Image 1 (opensbi)
  Description:  OpenSBI Dynamic Firmware
  Created:      Wed Mar  5 09:38:24 2025
  Type:         Firmware
  Compression:  uncompressed
  Data Size:    274920 Bytes = 268.48 KiB = 0.26 MiB
  Architecture: RISC-V
  OS:           RISC-V OpenSBI
  Load Address: 0x87000000
  Hash algo:    sha512
  Hash value:   67ade658ed95f0a6b731594c16bb853cdff431f8a739e67885b38967961bb312b3fbccfc4ec6c975628696c8b
ef5dd36a342c0bcf22f7330a6d8ecefce2b1a38
  Sign algo:    sha384,rsa3072:ventana_rsa-1
```

```
  Sign value:   321381e3f91ebb434be62fc3267022ec4fc27f84be043be9fa7c1220bc4eacf5b09d4c5efd1218406cc2594fa
4b0b27b6e7c50d2d37eab4db851e504b067ef6c17c23dab1ce72d1472d5d132a76d5f862486457845124a6c6620b2ee3837fbb41f
983494a5c26a21b5321dbd51ff84c14cc8fb98afd72f5d76126a0f2470dc0a806052ed344dfd0f79c8079901c9bd8901b76df60cd
19071bae1207cb0f34a1661d3cfb28e03d57d53113c1abfc2557424a2b6008342d0a12e87fd8c9c230c3e26b3462fa77b276abb27
bb3f0e549b4c8bf33605c15d36431c0bbaac5aa3bbeebbd75e9dba5bef7dad1fed571f59b830c770e53f88b1c56c6f3bef3b1cd8e
af71dc550295954bf9241e495451fc8c5b1da7fa0255dd36e172c8a4af4a6e1302a28e8ab563066a1dd8f0899bb8e70b3289d316d
c52bc2d1815f18974eae980786eb42f40c25e3b5c3e81c0edbe7baf43e4caedd653fe548c8d2e23b321cfa412aa544f1bac26ad64
9f314699e1961fa01a8b1c2021b761c214121eb2858978215
  Timestamp:    Wed Mar  5 09:55:40 2025
 Configuration 0 (conf-0)
  Description:  FIT Image and Signature Configuration
  Kernel:       unavailable
  Firmware:     opensbi
  Loadables:    edk2
  Sign algo:    sha384,rsa3072:ventana_rsa-0
  Sign value:   5fa8f2cf6e979f7b0e3590acf0ab5a4e146c19a45116d74d98aba716f11523b7ea0c4abd651dc9d73217372f0
2255153fdb16390b33d6063c5721433dcecb6353fb2a3e513433618dee9c432347418d57f58f18e787f4a8631f79f206512c96790
623fb763d73e8598fd91ffd1b3856e1a49c62a720e3cc3cde280759305133dcdd9e994bd7d73d85a71cf0b007c7d47ded988d9c36
0fa540a55e18617b0dea1062b9eefdec52cc68aafcd1d3fe99e476d78f2591f2b0e0ab3028ac594bac5e5197716519f03ee05c6bb
180dc3d55c25076a0d5e9599887c857ec72962b34b58ee8b2afe950ab1279b18440ff4282b284f494c8b70f53aa95bd37642ee385
d313bb3b095a5ed1cecd82babe10fd5e9d370fb579f44846c2ec25e5508815cf0350d5f30871ff2ad0bf7906aaa4abd623be90dac
1d7314306c241346da61db00175c628760986ffa575f2d90c1db3cea75e2b5b96ddebfd9b92a584b12c1f060cbace46033cbc8860
7cd278bd0fe637616e8aad59187a3e9c02e43720fb18796e3
  Timestamp:    Wed Mar  5 09:55:40 2025
Signature written to '/home/user/code/Releases/R-0.17.1/ventana-sw/platform/ventana-synth-vx/edk2/SFitIma
ge', node '/configurations/conf-0/signature-0'
Public key written to '/home/user/code/Releases/R-0.17.1/ventana-sw/platform/ventana-synth-vx/secure-boot
/keys/ventana_rsa-pubkeys.dtb', node '/signature/key-ventana_rsa-0'
```

Note : Screenshot is illustrative

As can be seen in the above sample output, now corresponding to all the **signature** nodes added in the **signfit-edk2.its** file, the **Sign value** field is filled with some 3072 bit value which means that the FIT image has been signed. In general, the size of the value in the **Sign value** field will be as per the **Sign algo** chosen.

The FIT public key dtb file **ventana_rsa-pubkeys.dtb** under the **$SBOOT_DIR/keys** folder at this point contains the final set of public keys corresponding to the private keys used for signing **images** and **configurations**. This file will need to be passed through the **FIT_PKDTB** parameter when signing the zStage image later so that it becomes a part of Chain of Trust.

Please ensure that the signing keys used in the **signfit-edk2.its** file actually exist in the path specified by the -k parameter and the following kind of message is not seen at all –

```
Couldn't open RSA private key: path_to_keys_folder/key-name-hint.key':
No such file or directory
```

In case, this gets missed and further usage is continued with the generated **FitImage**, it would lead to some signature verification failure at runtime followed by the message –

```
### ERROR ### Please RESET the board ###
```

8.3. Create Signed zStage Image/ Bundle

Only for **ventana-synth** platforms, booting a signed zStage image/ bundle is supported. Following set of parameters are required to sign a zStage image –

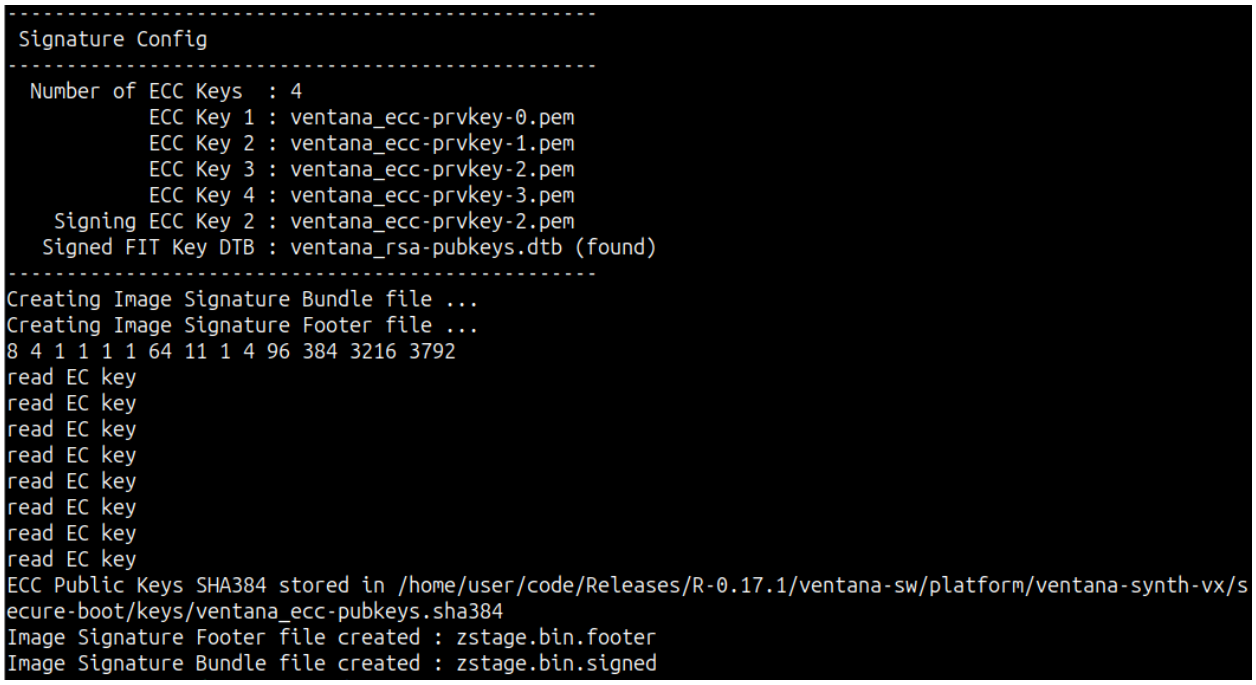
Parameter Name	Parameter details
-b	Path to the folder having the **ventana-synth** platform specific zStage image (i.e., **zstage.bin**) that is intended to be signed.
-p	Common path to the folder having the ECC keys and final FIT public key dtb filename to be mentioned wrt **-f** parameter.
-n	Common base prefix name of the ECC keys. E.g., wrt ECC keys **ventana_ecc-prvkey-x.pem** generated earlier, the prefix name to be specified would be **ventana_ecc.**
-c	Total number of the ECC keys generated for the purpose of image signature generation (and hence signature verification).
-s	Index of the ECC key to be used for signature generation. [Note: `0 <= value < (max count value specified by -c)`]
-f	Final FIT public key dtb file generated post FIT image signing. Note: This parameter requires only filename, not the filename with path as path MUST be the same as specified for **-p** parameter.

Signed zStage bundle creation example using signing **script**:

```
$  cd $WORK_DIR/ventana-sw/tools
$  ./gen_zstage_signed_bundle.sh -h (just to see script usage help)
$  ./gen_zstage_signed_bundle.sh \
   -b $WORK_DIR/ventana-sw/platform/ventana-synth-v2/zstage \
   -p $SBOOT_DIR/keys \
   -n ventana_ecc -c 4 -s 2 -f ventana_rsa-pubkeys.dtb
```

WARNING: If the path as specified by `-b` parameter already contains the released `zstage.bin.signed`, then it will get overwritten. Save the original copy if required.

Sample output of the last command:



Note: Screenshot is illustrative

The final result of a successful signing process is a `zstage.bin.signed` file (aka zStage signed bundle) that gets created at the path specified by `-b` parameter. To establish Root of Trust before authenticating zStage signed bundle, file `ventana_ecc-pubkeys.sha384` will be needed which is generated during the signing process under the **`$SBOOT_DIR/keys`** folder.

8.4. Boot with Signed zStage Image/ Bundle (Secure Boot)

End to end Secure Boot is ensured starting with the Secure Boot of zStage image, followed up with the Secure Boot of OpenSBI and EDK2 images.

zStage launch by QEMU happens **if and only if** it has passed all the Secure Boot checks. Once the zStage is launched, the now trusted FIT public keys get passed to U-Boot SPL, which it uses to perform the Secure Boot of OpenSBI and EDK2.

QEMU shipped as part of the release establishes the Root of Trust, performs the Revocation Mask checks and then initiates the bundle authentication wrt zStage signed bundle. Two QEMU **`ventana-synth-xx`** machine parameters namely, **`pkhash-efuse`** and **`pkrevm-efuse`**

are to be passed when launching the QEMU command to help it establish the Root of Trust and perform Revocation Mask checks.

Parameter Name	Parameter details
pkhash-efuse	Simulated e-fuse which contains the SHA384 value of the set of public keys of the platform owner to establish Root of Trust. The file **ventana_ecc-pubkeys.sha384** generated during zStage signing has to be passed through this parameter. The file contains the SHA384 value of the desired set of public keys.
pkrevm-efuse	Simulated e-fuse to revoke/ disable the use of a compromised set of keys. The value to be passed must be in decimal only. Let the total number of keys be 4. If only 1 key (say key 2) is to be revoked, then the value to be set should be **pkrevm-efuse=4** (0b0100 = 0x4 = 4) and if 2 keys are to be revoked (say key 1 and key 3), then the value will have to be **pkrevm-efuse=10** (0b1010 = 0xA = 10).

Run an emulated 64-bit Ventana Synth V2 machine using QEMU with the following command including the **pkhash-efuse** and **pkrevm-efuse** parameter:

```
$  $WORK_DIR/ventana-sw/QEMU-x86_64-Ubuntu-20.04.AppImage -M
   ventana-synth-v2,pkhash-efuse=$SBOOT_DIR/keys/ventana_ecc-pubkeys.sha384
   ,pkrevm-efuse=0 -nographic -bios
   $WORK_DIR/ventana-sw/platform/ventana-synth-v2/zstage/zstage.bin.signed
   -device virtio-net-pci,netdev=eth0 -netdev
   user,hostfwd=tcp::9991-:22,id=eth0 -drive
   file=$WORK_DIR/ventana-sw/platform/bios_edk2_riscv64_ventana.img,format=
   raw,if=sd -drive
   file=$WORK_DIR/ventana-sw/platform/ventana-synth-v2/ubuntu-25.04-preinst
   alled-server-riscv64+synth-v2.img,id=hd0,format=raw -device
   nvme,serial=ddaaddaa,drive=hd0
```

This **bios** parameter which contains the zStage image name (with/ without **.signed** suffix) inherently acts as a configuration switch to enable/ disable Secure Boot. When this

parameter specifies a signed image, it is treated as a request to enable Secure Boot and when it specifies an unsigned image, Secure Boot remains disabled. The system is expected to boot with Secure Boot disabled (no related checks performed) and when Secure Boot is enabled, the system is expected to boot only when all the due checks are successful.

```
QEMU-x86_64-Ubuntu-20.04.AppImage: info: Signed image: VMSAUTH library in use (v0.4)
QEMU-x86_64-Ubuntu-20.04.AppImage: info: Signed image: Root of Trust established (safe to use non-revoked ECC keys)
QEMU-x86_64-Ubuntu-20.04.AppImage: info: Signed image: None of the ECC Key(s) revoked
QEMU-x86_64-Ubuntu-20.04.AppImage: info: Signed image: zStage Bundle Authentication/ Signature Verification by VMSAUTH: SUCCESS !!!
QEMU-x86_64-Ubuntu-20.04.AppImage: info: Signed image: zStage Secure Boot ensured !!!

Ventana zStage v0.1
zstage: console device uart8250
zstage: setup platform FDT at 0x83f00000
zstage: copying next stage (130008 bytes) to 0x84000000 from 0x8001e000
zstage: next booting stage at 0x84000000 (130008 bytes)
zstage: harts 0x2*, 0x5, 0x6, 0x1, 0x4, 0x3, 0x0
Updating cache-size from 67108864 to 29360128
Updating cache-sets from 65536 to 28672
zstage: adding key subnode 'key-ventana_rsa-0' [required = "conf"]
zstage: adding key subnode 'key-ventana_rsa-1' [required = "image"]
zstage: adding key subnode 'key-ventana_rsa-2' [required = "image"]
zstage: Signed image: Secure Boot is enabled for next booting stage
zstage: jumping to next booting stage

U-Boot SPL 2025.01-rc3-00029-gd33caaeeb3c3 (Feb 26 2025 - 13:56:33 +0530)
Trying to boot from MMC1
## Checking signature(s) for config 'conf-0' ...
-- Check for signature wrt 'key-ventana_rsa-0' node [required = "conf"] ...
** Checking 'signature-0' node ... sha384,rsa3072:ventana_rsa-0+ ==========> OK
## Checking signature(s) and/or hash(es) for Image 'opensbi' ...
-- Check for signature wrt 'key-ventana_rsa-2' node [required = "image"] ...
-- Checking 'signature-1' node ... sha384,rsa3072:ventana_rsa-1+
-- Check for signature wrt 'key-ventana_rsa-1' node [required = "image"] ...
-- Checking 'signature-1' node ... sha384,rsa3072:ventana_rsa-1+ sha512+ ===> OK
## Checking signature(s) and/or hash(es) for Image 'edk2' ...
-- Check for signature wrt 'key-ventana_rsa-2' node [required = "image"] ...
-- Checking 'signature-2' node ... sha384,rsa3072:ventana_rsa-2+
-- Check for signature wrt 'key-ventana_rsa-1' node [required = "image"] ...
-- Checking 'signature-2' node ... sha384,rsa3072:ventana_rsa-2+ sha512+ ===> OK
WARNING: riscv,rpmi-system-reset driver is experimental and may change
WARNING: riscv,rpmi-shmem-mbox driver is experimental and may change
WARNING: riscv,rpmi-system-suspend driver is experimental and may change
WARNING: riscv,rpmi-hsm driver is experimental and may change
WARNING: riscv,rpmi-cppc driver is experimental and may change

OpenSBI v1.6-62-g48a78951
   ____                    ____ ____ _____
  / __ \                  / ___|  _ \_   _|
 | |  | |_ __   ___ _ __ | (___ | |_) || |
 | |  | | '_ \ / _ \ '_ \ \___ \|  _ < | |
 | |__| | |_) |  __/ | | |____) | |_) || |_
  \____/| .__/ \___|_| |_|_____/|____/_____|
        | |
        |_|
```

...

```
RISC-V EDK2 firmware version stable202502
Press ESCAPE within 10 seconds for boot options
UEFI Interactive Shell v2.2
EDK II
UEFI v2.70 (EDK II, 0x00010000)
map: No mapping found.
Press ESC in 1 seconds to skip startup.nsh or any other key to continue.
Shell>
```

Note: Screenshots are illustrative

9. OpenBMC support

This release is being shipped with all necessary components having the desired changes / configuration to enable the **OpenBMC support** feature.

Note:- This feature is supported ONLY on **Ventana Synth** platforms (all variants).

In order to set this feature up, QEMU images for both riscv64 and arm platforms need to be built along with OpenBMC flash image. Once BMC and Synth host machines are up, BMC will start responding to accesses from serial, web and redfish/IPMI clients.

Following sections will first describe the steps to build components, boot the QEMU machines and then verification methods.

9.1. Extract Ventana pre-built OpenBMC binaries

Extract Ventana OpenBMC binaries.

```
$ export WORK_DIR=`pwd` && cd $WORK_DIR
$ mkdir ventana-openbmc
$ tar -C ventana-openbmc -xf
ventana-sw-openbmc-${VENTANA_SW_VERSION}.tar.xz
```

9.2. Generate OpenBMC flash image

Build Ventana BMC flash image:

```
$ git clone -b release-v0.18.1 git@github.com:ventanamicro-dev/openbmc.git
$ cd openbmc
$ . setup vttunga
```

```
$ export LC_ALL=C
$ bitbake obmc-phosphor-image
$ ls
$WORK_DIR/openbmc/build/vttunga/tmp/deploy/images/vttunga/obmc-phosphor-ima
ge-vttunga.static.mtd
```

9.3. Boot OpenBMC with Ventana Synth host

Boot Ventana Synth machine:

```
$ cd $WORK_DIR
$ $WORK_DIR/ventana-sw/QEMU-x86_64-Ubuntu-20.04.AppImage -M
ventana-synth-v2,ri2c=on -smp 2 -m 8G -nographic \
-bios $WORK_DIR/ventana-sw/platform/ventana-synth-v2/zstage/zstage.bin \
-device virtio-net-pci,netdev=eth0,romfile= -netdev
user,hostfwd=tcp::2299-:22,id=eth0 \
-drive
file=$WORK_DIR/ventana-sw/platform/bios_edk2_riscv64_ventana.img,format=raw
,if=sd \
-drive
file=$WORK_DIR/ventana-sw/platform/ubuntu-25.04-preinstalled-server-riscv6.
img,id=hd0,format=raw \
-device nvme,serial=ddaaddaa,drive=hd0 \
-monitor telnet:127.0.0.1:4444,server,nowait \
-serial telnet:127.0.0.1:45457,server,nowait
```

Boot BMC machine:

```
$WORK_DIR/ventana-openbmc/QEMU-x86_64-Ubuntu-20.04.AppImage -m 512M -M
ventana-bmc -nographic \
-drive
file=$WORK_DIR/ventana-openbmc/obmc-phosphor-image-vttunga.static.mtd,forma
t=raw,if=mtd \
-device
remote-i2c,bus=aspeed.i2c.bus.1,id=remote-i2c-master,address=0x62,tsocket=/
tmp/master-socket \
```

```
-net nic -net
user,hostfwd=:127.0.0.1:2222-:22,hostfwd=:127.0.0.1:2443-:443,hostfwd=tcp:1
27.0.0.1:2200-:2200,hostfwd=udp:127.0.0.1:2623-:623 \
-serial tcp::45451,server,nowait -serial tcp::45452,server,nowait -serial
chardev:ch0 -chardev
socket,id=ch0,host=0.0.0.0,port=45457,telnet=on,server=off
```

9.4. Verify BMC ready on console

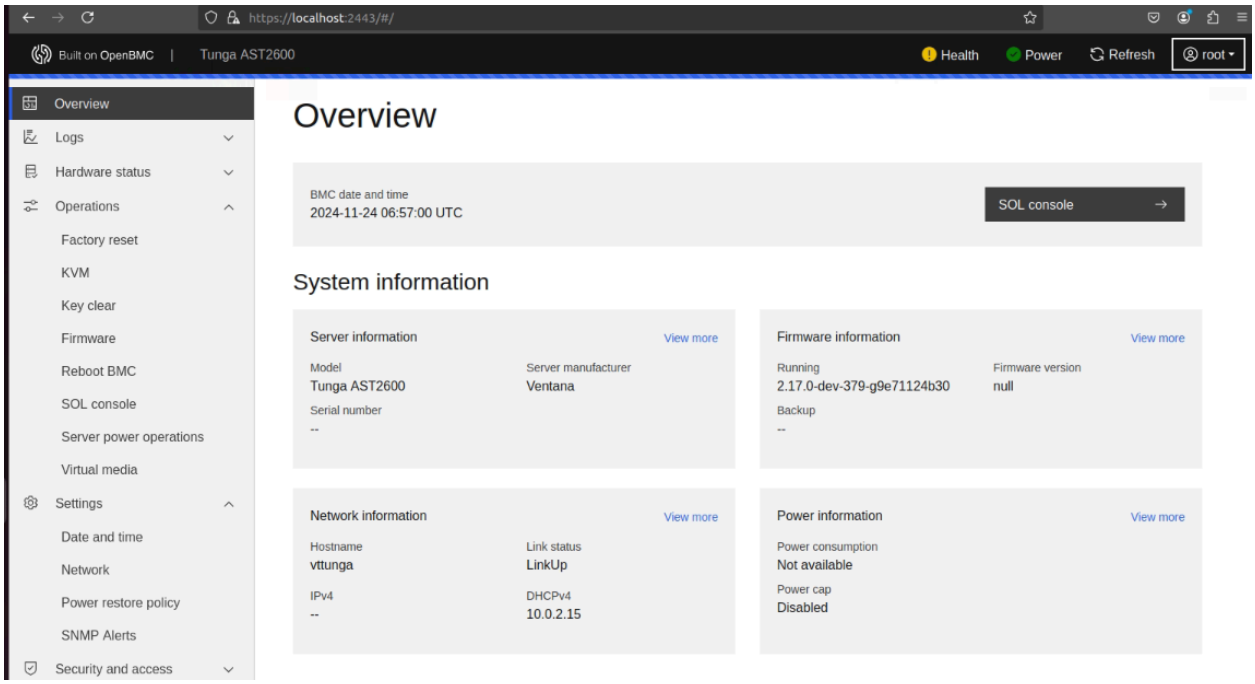
```
$ cd $WORK_DIR
$ telnet localhost 45451
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.


U-Boot 2019.04 (Jul 26 2024 - 07:18:46 +0000)

.
.
$ obmcutil state
CurrentBMCState:    xyz.openbmc_project.State.BMC.BMCState.Ready
CurrentPowerState:  xyz.openbmc_project.State.Chassis.PowerState.On
CurrentHostState:   xyz.openbmc_project.State.Host.HostState.Running
```

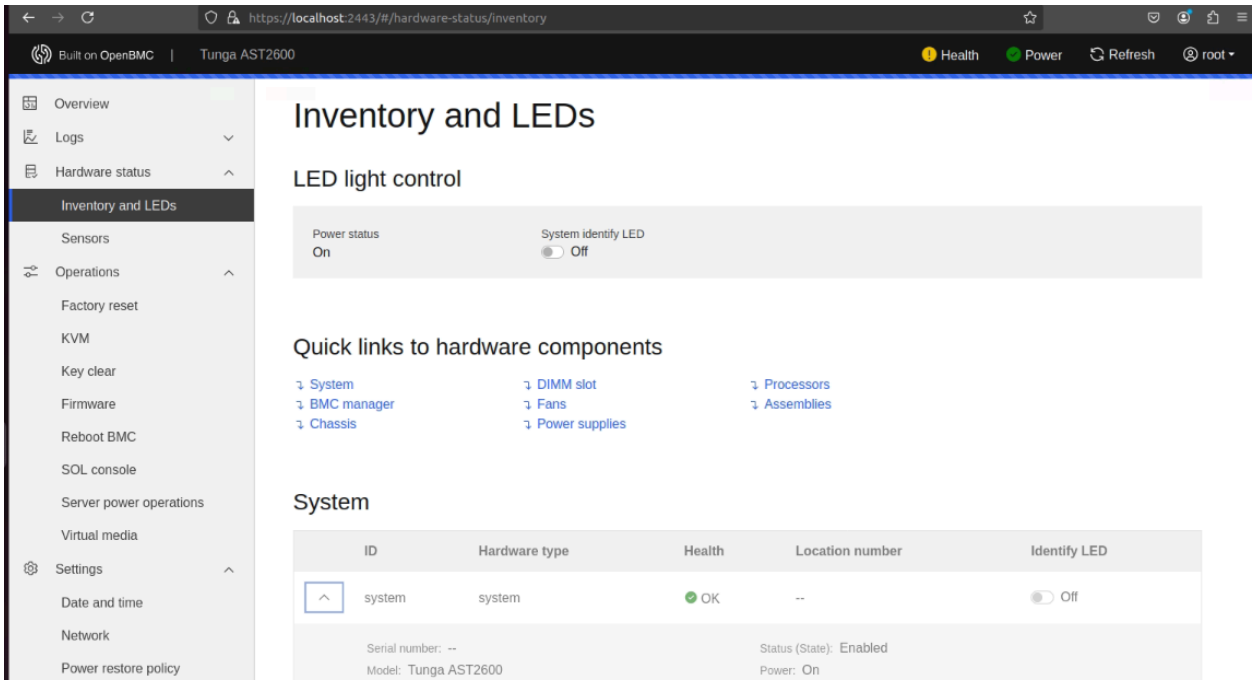
9.5. Verify BMC Web GUI

Once the BMC has booted, connect to web UI and login https://localhost:2443

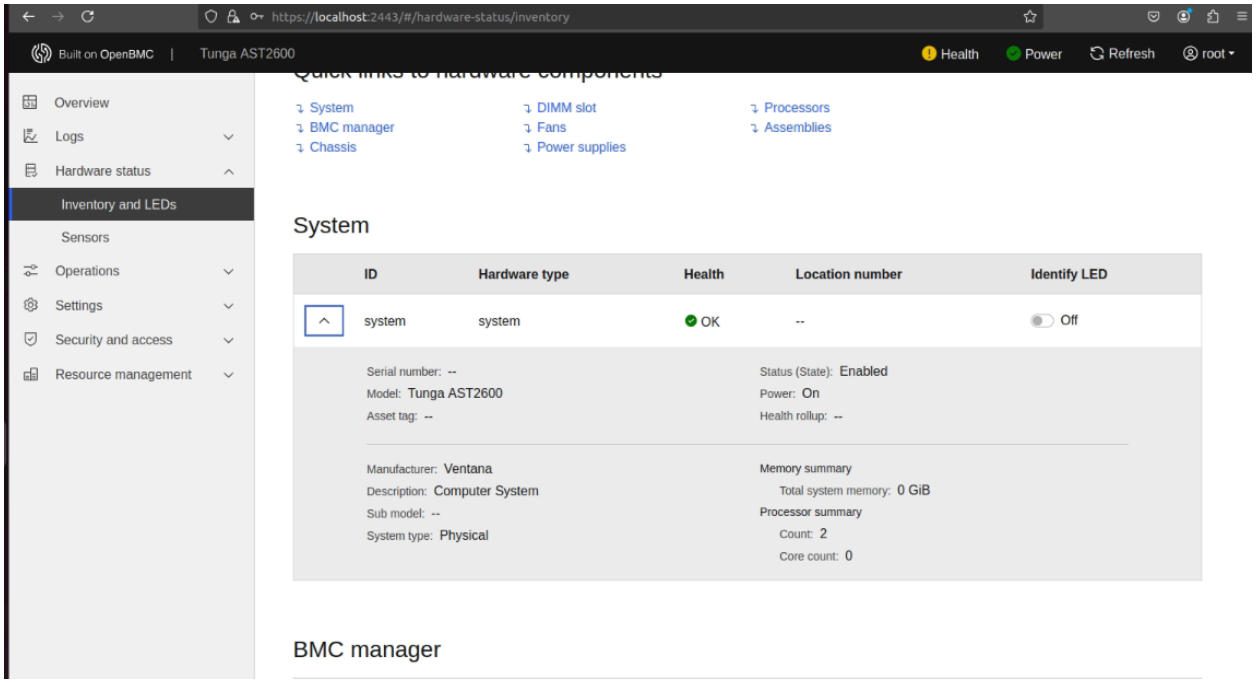


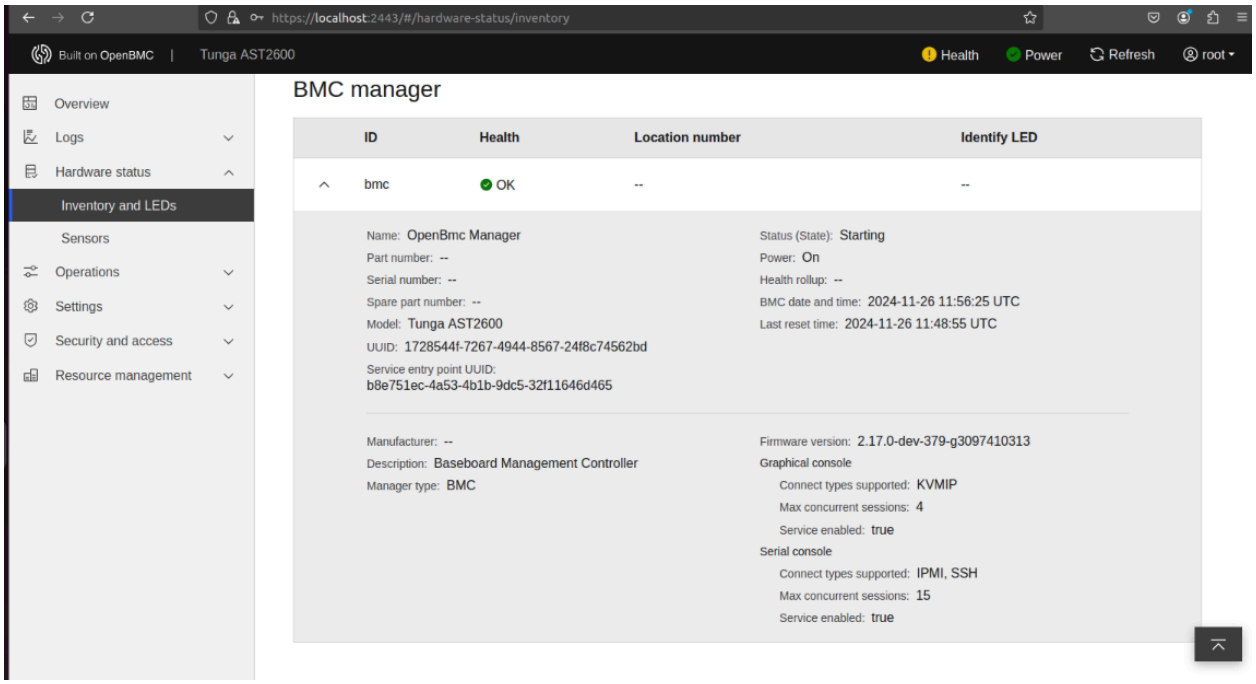
9.6. Verify BMC inventory

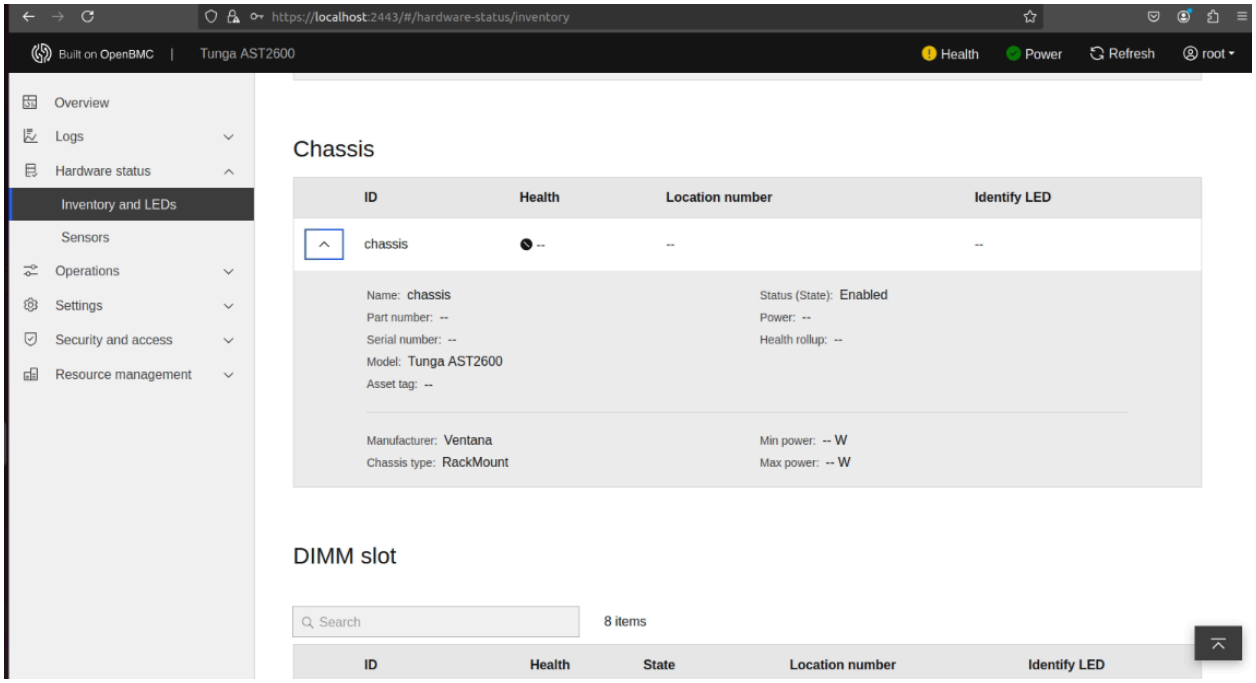
Click on Hardware status -> Inventory and LEDs

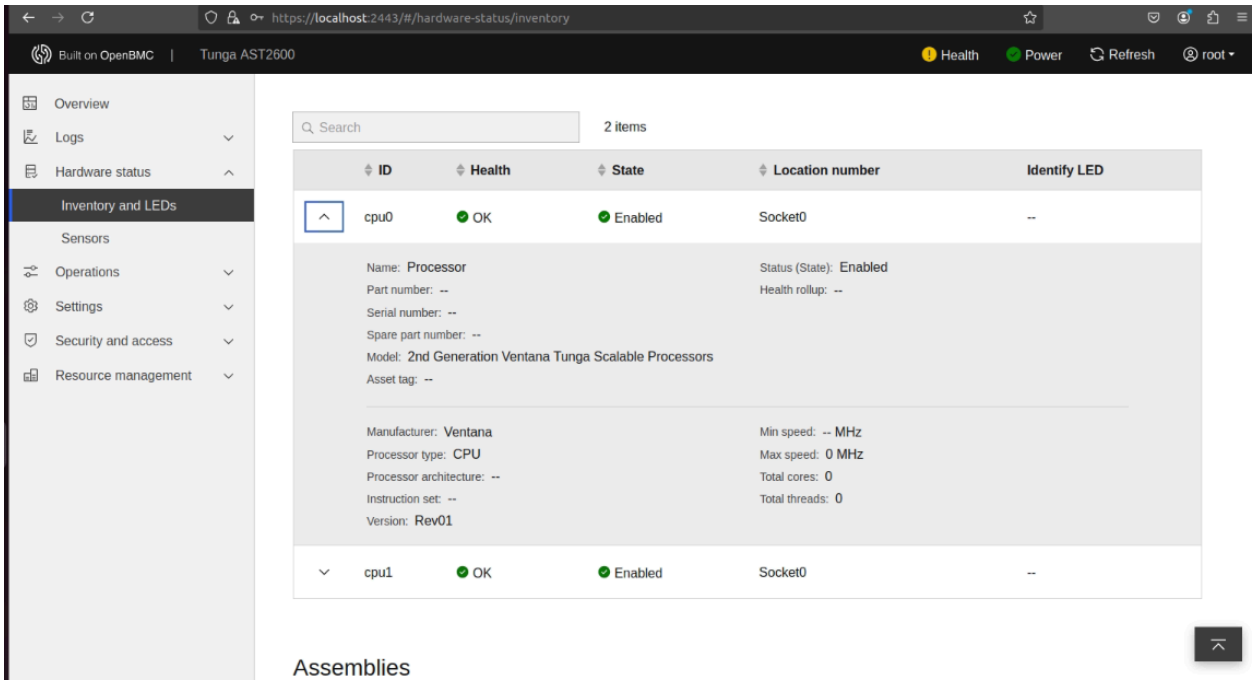


Scroll down, expand System, BMC sections to see Ventana specific details.




https://localhost:2443/#/hardware-status/inventory
Built on OpenBMC | Tunga AST2600
Health
Power
Refresh
root
Overview
Logs
Hardware status
Inventory and LEDs
Sensors
Operations
Settings
Security and access
Resource management
BMC manager
ID
Health
Location number
Identify LED
bmc
OK
--
--
Name: OpenBmc Manager
Part number: --
Serial number: --
Spare part number: --
Model: Tunga AST2600
UUID: 1728544f-7267-4944-8567-24f8c74562bd
Service entry point UUID:
b8e751ec-4a53-4b1b-9dc5-32f11646d465
Status (State): Starting
Power: On
Health rollup: --
BMC date and time: 2024-11-26 11:56:25 UTC
Last reset time: 2024-11-26 11:48:55 UTC
Manufacturer: --
Description: Baseboard Management Controller
Manager type: BMC
Firmware version: 2.17.0-dev-379-g3097410313
Graphical console
Connect types supported: KVMIP
Max concurrent sessions: 4
Service enabled: true
Serial console
Connect types supported: IPMI, SSH
Max concurrent sessions: 15
Service enabled: true


https://localhost:2443/#/hardware-status/inventory
Built on OpenBMC | Tunga AST2600
Health
Power
Refresh
root
Overview
Logs
Hardware status
Inventory and LEDs
Sensors
Operations
Settings
Security and access
Resource management
Chassis
ID
Health
Location number
Identify LED
chassis
--
--
Name: chassis
Part number: --
Serial number: --
Model: Tunga AST2600
Asset tag: --
Status (State): Enabled
Power: --
Health rollup: --
Manufacturer: Ventana
Chassis type: RackMount
Min power: -- W
Max power: -- W
DIMM slot
Search
8 items
ID
Health
State
Location number
Identify LED



9.7. Verify BMC Serial over LAN

Click on Operations->SOL/ Serial over LAN Console and connect to host serial machine with credentials:
User: ventana
Password: ventana

Click Server power options -> Immediate Reboot to reset the Synth host, and watch SOL console to watch host rebooting logs.

9.8. Verify BMC IPMI connection from client

Verify BMC using ipmitool:

```
CLIENT $ ipmitool -I lanplus  -C 17  -H 127.0.0.1 -p 2623 -P 0penBmc -U
root chassis status
System Power         : on
```

```
Power Overload       : false
Power Interlock      : inactive
Main Power Fault     : false
Power Control Fault  : false
Power Restore Policy : always-on
Last Power Event     :
Chassis Intrusion    : inactive
Front-Panel Lockout  : inactive
Drive Fault          : false
Cooling/Fan Fault    : false
Sleep Button Disable : not allowed
Diag Button Disable  : not allowed
Reset Button Disable : allowed
Power Button Disable : allowed
Sleep Button Disabled: false
Diag Button Disabled : false
Reset Button Disabled: false
Power Button Disabled: false

$ ipmitool -I lanplus  -C 17  -H 127.0.0.1 -p 2623 -P 0penBmc -U root mc
info
Device ID                 : 32
Device Revision           : 1
Firmware Revision         : 2.17
IPMI Version              : 2.0
Manufacturer ID           : 52538
Manufacturer Name         : Unknown (0xCD3A)
Product ID                : 131 (0x0083)
Product Name              : Unknown (0x83)
Device Available          : yes
Provides Device SDRs      : no
Additional Device Support :
    Sensor Device
    SEL Device
    FRU Inventory Device
    Chassis Device
Aux Firmware Rev Info     :
    0x00
    0x00
```

```
    0x00
    0x00

$ ipmitool -I lanplus  -C 17  -H 127.0.0.1 -p 2623 -P 0penBmc -U root sol
info
Info: SOL parameter 'Volatile Bitrate (6)' not supported
Set in progress                 : set-complete
Enabled                         : true
Force Encryption                : true
Force Authentication            : true
Privilege Level                 : ADMINISTRATOR
Character Accumulate Level (ms) : 100
Character Send Threshold        : 1
Retry Count                     : 7
Retry Interval (ms)             : 1000
Volatile Bit Rate (kbps)        : IPMI-Over-Serial-Setting
Non-Volatile Bit Rate (kbps)    : IPMI-Over-Serial-Setting
Payload Channel                 : 1 (0x01)
Payload Port                    : 623
```

9.9. Verify BMC redfish connection from client

```
$ cd $WORK_DIR
$ export bmc=localhost:2443
$ export token=`curl -k -H "Content-Type: application/json" -X POST
https://${bmc}/login -d '{"username" :  "root", "password" :  "0penBmc"}' |
grep token | awk '{print $2;}' | tr -d '"'`

$ curl -k -H "X-Auth-Token: $token" -X GET
https://${bmc}/redfish/v1/Systems
{
  "@odata.id": "/redfish/v1/Systems",
  "@odata.type": "#ComputerSystemCollection.ComputerSystemCollection",
  "Members": [
```

```
    {
      "@odata.id": "/redfish/v1/Systems/system"
    }
  ],
  "Members@odata.count": 1,
  "Name": "Computer System Collection"
}r

$ curl -u root:0penBmc -k -s  https://${bmc}/redfish/v1/Systems/system
{
  "@odata.id": "/redfish/v1/Systems/system",
  "@odata.type": "#ComputerSystem.v1_22_0.ComputerSystem",
  "Actions": {
    "#ComputerSystem.Reset": {
      "@Redfish.ActionInfo": "/redfish/v1/Systems/system/ResetActionInfo",
      "target": "/redfish/v1/Systems/system/Actions/ComputerSystem.Reset"
    }
  },
  "Bios": {
    "@odata.id": "/redfish/v1/Systems/system/Bios"
  },
  "BiosVersion": "null",
  "Boot": {
    "AutomaticRetryAttempts": 3,
    "AutomaticRetryConfig": "RetryAttempts",
    "AutomaticRetryConfig@Redfish.AllowableValues": [
      "Disabled",
      "RetryAttempts"
    ],
    "BootSourceOverrideEnabled": "Disabled",
    "BootSourceOverrideMode": "UEFI",
    "BootSourceOverrideMode@Redfish.AllowableValues": [
      "Legacy",
      "UEFI"
    ],
    "BootSourceOverrideTarget": "None",
    "BootSourceOverrideTarget@Redfish.AllowableValues": [
      "None",
      "Pxe",
```

```
      "Hdd",
      "Cd",
      "Diags",
      "BiosSetup",
      "Usb"
    ],
    "RemainingAutomaticRetryAttempts": 3,
    "StopBootOnFault": "Never",
    "TrustedModuleRequiredToBoot": "Disabled"
  },
  "BootProgress": {
    "LastState": "OSRunning",
    "LastStateTime": "1970-01-01T00:00:00.000000+00:00"
  },
  "Description": "Computer System",
  "FabricAdapters": {
    "@odata.id": "/redfish/v1/Systems/system/FabricAdapters"
  },
  "GraphicalConsole": {
    "ConnectTypesSupported": [
      "KVMIP"
    ],
    "MaxConcurrentSessions": 4,
    "ServiceEnabled": true
  },
  "Id": "system",
  "LastResetTime": "1970-01-01T00:00:00+00:00",
  "Links": {
    "Chassis": [
      {
        "@odata.id": "/redfish/v1/Chassis/chassis"
      }
    ],
    "ManagedBy": [
      {
        "@odata.id": "/redfish/v1/Managers/bmc"
      }
    ]
  },
```

```
"LogServices": {
  "@odata.id": "/redfish/v1/Systems/system/LogServices"
},
"Manufacturer": "Ventana",
"Memory": {
  "@odata.id": "/redfish/v1/Systems/system/Memory"
},
"MemorySummary": {
  "TotalSystemMemoryGiB": 0.0
},
"Model": "Tunga AST2600",
"Name": "system",
"PCIeDevices": [],
"PCIeDevices@odata.count": 0,
"PartNumber": "",
"PowerRestorePolicy": "AlwaysOn",
"PowerState": "On",
"ProcessorSummary": {
  "CoreCount": 0,
  "Count": 2
},
"Processors": {
  "@odata.id": "/redfish/v1/Systems/system/Processors"
},
"SerialConsole": {
  "IPMI": {
    "ServiceEnabled": true
  },
  "MaxConcurrentSessions": 15,
  "SSH": {
    "HotKeySequenceDisplay": "Press ~. to exit console",
    "Port": 2200,
    "ServiceEnabled": true
  }
},
"SerialNumber": "",
"Status": {
  "Health": "OK",
  "State": "Enabled"
```

```
  },
  "Storage": {
    "@odata.id": "/redfish/v1/Systems/system/Storage"
  },
  "SubModel": "",
  "SystemType": "Physical"
}

$ curl -u root:0penBmc -k -s
https://${bmc}/redfish/v1/Systems/system/Processors
{
  "@odata.id": "/redfish/v1/Systems/system/Processors",
  "@odata.type": "#ProcessorCollection.ProcessorCollection",
  "Members": [
    {
      "@odata.id": "/redfish/v1/Systems/system/Processors/cpu0"
    },
    {
      "@odata.id": "/redfish/v1/Systems/system/Processors/cpu1"
    }
  ],
  "Members@odata.count": 2,
  "Name": "Processor Collection"
}

$ curl -u root:0penBmc -k -s
https://${bmc}/redfish/v1/Systems/system/Processors/cpu0
{
  "@odata.id": "/redfish/v1/Systems/system/Processors/cpu0",
  "@odata.type": "#Processor.v1_18_0.Processor",
  "Id": "cpu0",
  "Location": {
    "PartLocation": {
      "ServiceLabel": "Socket0"
    }
  },
  "Manufacturer": "Ventana",
  "MaxSpeedMHz": 0,
  "Model": "2nd Generation Ventana Tunga Scalable Processors",
```

```
  "Name": "Processor",
  "PartNumber": "",
  "ProcessorType": "CPU",
  "Socket": "",
  "Status": {
    "Health": "OK",
    "State": "Enabled"
  },
  "TotalCores": 0,
  "TotalThreads": 0,
  "Version": "Rev01"
}
```


10. RAS Support

RAS (Reliability, Availability and Serviceability) features are design elements and techniques used in computer systems to enhance their dependability and operational efficiency. These features aim to minimize downtime, ensure data integrity, and facilitate easy maintenance and repair.

The Ventana SDK from v0.18.1 onwards supports RAS.

10.1. Verify the support for RAS

RAS builds on hardware's capability to report errors to software. The software then takes appropriate actions. The RAS is implemented as per the ACPI 6.5 Specification.

The keyword "HEST" can be searched in the Linux boot log to see if RAS is supported.

```
ventana@ventana:~$ dmesg | grep HEST
[    0.000000] ACPI: HEST 0x00000000FF11C098 00041C (v02 VNTANA VENTANA  00000002 VNTN 00000001)
[    1.323240] HEST: Table parsing has been initialized.
```

The above log shows that RAS is supported and it is initialized.

NOTE: For the curious mind, HEST stands for Hardware Error Source Table - An ACPI defined way of reporting hardware that can report errors and ways to do it.

10.2. ACPI EINJ

ACPI EINJ provides a hardware error injection mechanism. It is very useful for testing RAS features.

It needs to be checked whether EINJ is supported. For that, look for early boot messages similar to this one:

```
ventana@ventana:~$ dmesg | grep EINJ
[    0.000000] ACPI: EINJ 0x00000000FF0DD098 000150 (v02 VNTANA VENTANA  00000002 VNTN 00000001)
[    2.337656] EINJ: Error INJection is initialized.
```

As can be seen in the above log, the EINJ table was found and its driver in Linux was initialized. The absence of such messages might indicate that the EINJ stack in the Linux kernel is not enabled or not supported by the current kernel configuration.

10.3. Injecting an Error and Testing RAS

Let's inject the "Processor Correctable" error using the EINJ framework. Note that the number assigned to is **0x00000001**. First, set the error in "error_type" file:

root@ventana:/sys/kernel/debug/apei/einj# echo 0x00000001 > error_type

Next, set the hart ID in "param1" file. The indexing starts from 1. So, for hart 0, 1 should be written to the param1 file.

root@ventana:/sys/kernel/debug/apei/einj# echo 1 > param1

Set the "notrigger" to 0. So that error is triggered after injection.

root@ventana:/sys/kernel/debug/apei/einj# echo 0 > notrigger

Then, inject the error:

root@ventana:/sys/kernel/debug/apei/einj# echo 1 > error_inject

The following output can be seen on the screen showing that EINJ framework and the RAS framework both work.

```
root@ventana:/sys/kernel/debug/apei/einj# [19211.736605] {1}[Hardware Error]: Hardware error from APEI Generic Hardware Error Source: 1
[19211.741324] {1}[Hardware Error]: event severity: recoverable
[19211.741709] {1}[Hardware Error]:  Error 0, type: recoverable
[19211.742024] {1}[Hardware Error]:   section_type: general processor error
[19211.742374] {1}[Hardware Error]:   processor_type: 3, RISCV
[19211.742626] {1}[Hardware Error]:   processor_isa: 6, RISCV64
[19211.742949] {1}[Hardware Error]:   error_type: 0x08
[19211.743453] {1}[Hardware Error]:   micro-architectural error
[19211.743852] {1}[Hardware Error]:   operation: 3, instruction execution
```

NOTE: RAS and EINJ are in the early development phase. Only a few error types are supported right now. More errors and their handler will be added in future versions.

More details on EINJ support in Linux kernel can be found at
https://docs.kernel.org/firmware-guide/acpi/apei/einj.html

11. Appendix A. Revision history

Revision	Date	Comments
Rev 0.1.1	23-April-2021	• Initial revision
Rev 0.2.1	30-June-2021	• UEFI • PMU
Rev 0.3.1	30-Sep-2021	• HiFive Unmatched board support • Ventana RootFS • LLVM support

Rev 0.3.2		• PMU support updated
Rev 0.4.1	31-Dec-2021	• Added Ventana machine (ventana-vt1) with Ventana RV64 chiplet
Rec 0.5.1	31-Mar-2022	• zStage (new first stage loader) • EDK II S-mode
Rev 0.6.1	30-June-2022	• Added initial support of Ventana V1 perf events in Linux • Perf tool supports all Ventana V1 events
Rev 0.6.2	29-July-2022	• No updates in current document content
Rev 0.6.3	18-Aug-2022	• No updates in current document content
Rev 0.6.4	05-Sep-2022	• No updates in current document content
Rev 0.7.1	30-Sep-2022	• No updates in current document content
Rev 0.8.1	30-Dec-2022	• Moved to unified names for pre-built images ○ ventana-sw (Default images targeting QEMU and real world evaluation platforms) • Added SD card creation and flashing instructions for SiFive Unmatched board
Rev 0.8.2	06-Mar-2023	• No updates in current document content
Rev 0.9.1	31-Mar-2023	• No updates in current document content
Rev 0.10.1	30-June-2023	• Added Create rootfs section and Ventana Thunderhill board support
Rev 0.10.2	27-July-2023	• No updates in current document content
Rev 0.11.1	30-Sep-2023	• Updated steps to install bootloader itb files
Rev 0.12.1	05-Jan-2024	• Updated Ventana toolchain section • Updated Virtualization section • Updated SiFive HiFive Unmatched board SD card image steps • Added new platforms Ventana Veyron

		Thunderhill-v1.
Rev 0.13.1	31-Mar-2024	• Added new platforms ventana-synth-v1 and ventana-synth-v2.
Rev 0.14.1	30-Jun-2024	• Added command to build u-boot, zStage and EDK2 for ventana-thunderhill-v1 platform
Rev 0.14.2	05-Aug-2024	• No updates in current document content
Rev 0.15.1	30-Sep-2024	• No updates in current document content
Rev 0.16.1	07-Jan-2025	• Added command for secure boot • Added command for OpenBMC
Rev 0.17.1	31-Mar-2025	• Added command for zstage signed image
Rev 0.17.2	10-April-2025	• No updates in current document content
Rev 0.18.1	30-Jun-2025	• Removed Ventana vt1 • Removed HiFive unmatched Board • Added Ventana cloud init • Added RAS Support