Project Report
on
Linux Bootloaders using QEMU for RISC-V

Submitted
In partial fulfilment
For the award of the Degree of

PG-Diploma in Embedded Systems and Design (PG-DESD)

CDAC, ACTS (Pune)

Guided By:
Mr. Rushikesh Jadhav

Submitted By:
Jitesh Singh Chauhan (PRN: 230940130031)
Chandan Yadav (PRN: 230940130019)
Ramiti Joshi (PRN: 230940130045)
Sayali Yashwant Mahadik (PRN: 230940130056)

Centre for Development of Advanced Computing (CDAC), ACTS
Pune - 411008

Acknowledgement

This is to acknowledge our indebtedness to our Project Guide, **Mr Rushikesh Jadhav** C-DAC ACTS, Pune for her constant guidance and helpful suggestion for preparing this project **Linux Bootloaders using QEMU for RISC-V.** We express our deep gratitude towards her for inspiration, personal involvement, constructive criticism that she provided us along with technical guidance during the course of this project.

We take this opportunity to thank Head of the department **Mr. Gaur Sunder** for providing us such a great infrastructure and environment for our overall development.

We express sincere thanks to **Ms. Namrata Ailawar**, Process Owner, for their kind cooperation and extendible support towards the completion of our project.

It is our great pleasure in expressing sincere and deep gratitude towards **Ms. Risha P R (Program Head)** and **Ms. Srujana Bhamidi** (Course Coordinator, PG-DESD) for their valuable guidance and constant support throughout this work and help to pursue additional studies.

Also, our warm thanks to **C-DAC ACTS Pune**, which provided us this opportunity to carry out this prestigious Project and enhance our learning in various technical fields.

Jitesh Singh Chauhan (PRN: 230940130031)
Chandan Yadav (PRN: 230940130019)
Ramiti Joshi (PRN: 230940130045)
Sayali Yashwant Mahadik (PRN: 230940130056)

Table of Contents

Chapter 1
Introduction to Bootloaders

1.1 Introduction

A bootloader, also spelled boot loader or bootstrap loader, is a small program that is responsible for starting up a computer and loading the operating system. It is the first program that runs when a computer is turned on or restarted. The bootloader is essential for the proper functioning of a computer, as it allows the operating system to interact with the hardware and other software programs.

1.2 Main functions of a bootloader

Performs hardware initialization: The bootloader checks the basic hardware components of the computer, such as the CPU, memory, and storage devices, to ensure that they are functioning properly.

Loads the operating system: The bootloader locates the operating system kernel on the storage device and loads it into memory.

Transfers control to the operating system: Once the operating system kernel is loaded, the bootloader transfers control to it, and the operating system takes over the boot process.

1.3 Types of bootloaders

Stage 1 bootloader: This is a very small program that is stored in firmware on the computer's motherboard. It is responsible for loading the stage 2 bootloader.

Stage 2 bootloader: This is a larger program that is stored on the computer's storage device. It is responsible for loading the operating system kernel.

In addition to its basic functions, a bootloader can also perform other tasks, such as:

Providing a boot menu: This allows the user to select which operating system to boot from, if multiple operating systems are installed on the computer.

Updating the firmware: The bootloader can be used to update the firmware on the computer's motherboard.

Performing diagnostics: The bootloader can be used to diagnose hardware problems.

1.4 Examples of Bootloaders

- Extensible Firmware Interface(EFI) bootloader – Bootloader used in current windows machines
- Coreboot - Used in Google Chromebook
- U-boot - popular bootloader for embedded systems
- Bootmgr – Bootloader used in windows vista
- NT Loader (NTLDR) – Bootloader used in windows XP
- BootX – Bootloader used in MAC machines.
- Grand Unified Bootloader (GRUB) – Bootloader which is used in Unix-like machines. E.g., Linux
- AndroidBoot (Aboot) – Bootloader which is used in android.
- iBoot – Bootloader which is used in iPhones.

Chapter 2
RISC-V : Reduced Instruction Set Computer -V

2.1 Introduction to RISC-V

RISC-V [1] is an open standard instruction set architecture (ISA) based on established reduced instruction set computer (RISC) principles. Unlike most other ISA designs, RISC-V is provided under royalty-free open-source licences. Many companies are offering or have announced RISC-V hardware; open source operating systems with RISC-V support are available, and the instruction set is supported in several popular software toolchains.

As a RISC architecture, the RISC-V ISA is a load–store architecture. Its floating-point instructions use IEEE 754 floating-point. Notable features of the RISC-V ISA include: instruction bit field locations chosen to simplify the use of multiplexers in a CPU,[2] a design that is architecturally neutral, and a fixed location for the sign bit of immediate values to speed up sign extension.[3]

The instruction set is designed for a wide range of uses. The base instruction set has a fixed length of 32-bit naturally aligned instructions, and the ISA supports variable length extensions where each instruction can be any number of 16-bit parcels in length.[4] Extensions support small embedded systems, personal computers, supercomputers with vector processors, and warehouse-scale parallel computers.

The instruction set specification defines 32-bit and 64-bit address space variants. The specification includes a description of a 128-bit flat address space variant, as an extrapolation of 32- and 64-bit variants, but the 128-bit ISA remains "not frozen" intentionally, because as of 2023, there is so little practical experience with such large memory systems.[5]

Unlike other academic designs which are typically optimised only for simplicity of exposition, the designers intended that the RISC-V instruction set be usable for practical computers. As of June 2019, version 2.2 of the user-space ISA[6] and version 1.11 of the privileged ISA[6] are frozen, permitting software and hardware development to proceed. The user-space ISA, now renamed the Unprivileged ISA, was updated, ratified and frozen as version 20191213.[7] An external debug specification is available as a draft, version 0.13.2.[8]

The project began in 2010 at the University of California, Berkeley. There are now members in over 70 countries contributing and collaborating to define RISC-V open

specifications. RISC-V International, the non-profit managing RISC-V, is currently headquartered in Switzerland.[9]

2.2 History

The term RISC dates from about 1980. Before then, there was some knowledge (see John Cocke) that simpler computers can be effective, but the design principles were not widely described. Simple, effective computers have always been of academic interest, and resulted in the RISC instruction set DLX for the first edition of Computer Architecture: A Quantitative Approach in 1990 of which David Patterson was a co-author, and he later participated in the RISC-V origination. DLX was intended for educational use; academics and hobbyists implemented it using field-programmable gate arrays (FPGA), but it was never truly intended for commercial deployment. ARM CPUs, versions 2 and earlier, had a public-domain instruction set and are still supported by the GNU Compiler Collection (GCC), a popular free-software compiler. Three open-source cores exist for this ISA, but were never manufactured.

OpenRISC is an open-source ISA based on DLX, with associated RISC designs, and is fully supported with GCC and Linux implementations, although it too has few commercial implementations.

Krste Asanović at the University of California, Berkeley, had a research requirement for an open-source computer system, and in 2010, he decided to develop and publish one in a "short, three-month project over the summer" with several of his graduate students. The plan was to aid both academic and industrial users. David Patterson at Berkeley joined the collaboration as he was the originator of the Berkeley RISC, and the RISC-V is the eponymous fifth generation of his long series of cooperative RISC-based research projects at the University of California, Berkeley (RISC-I and RISC-II published in 1981 by Patterson, who refers to the SOAR architecture from 1984 as "RISC-III" and the SPUR architecture from 1988 as "RISC-IV"). At this stage, students provided initial software, simulations, and CPU designs.

First Raven1 brought up ST28nm at Berkeley Wireless Research Center (BWRC) June 2012. The RISC-V authors and their institution originally sourced the ISA documents and several CPU designs under BSD licenses, which allow derivative works—such as RISC-V chip designs—to be either open and free, or closed and proprietary. The ISA specification itself (i.e., the encoding of the instruction set) was published in 2011 as open source, with all rights reserved. The actual technical report (an expression of the specification) was later placed under a Creative Commons license to permit enhancement by external contributors through the RISC-V Foundation, and later RISC-V International.

2.3 Rationale

CPU design requires design expertise in several specialties: electronic digital logic, compilers, and operating systems. To cover the costs of such a team, commercial vendors of processor intellectual property (IP), such as Arm Ltd. and MIPS Technologies, charge royalties for the use of their designs, patents and copyrights. They also often require non-disclosure agreements before releasing documents that describe their designs' detailed advantages. In many cases, they never describe the reasons for their design choices.

RISC-V was begun with a goal to make a practical ISA that was open-sourced, usable academically, and deployable in any hardware or software design without royalties. Also, justifying rationales for each design decision of the project are explained, at least in broad terms. The RISC-V authors are academics who have substantial experience in computer design, and the RISC-V ISA is a direct development from a series of academic computer-design projects, especially Berkeley RISC. RISC-V was created in part to aid all such projects.

To build a large, continuing community of users and thereby accumulate designs and software, the RISC-V ISA designers intentionally support a wide variety of practical use cases: compact, performance, and low-power real-world implementations without over-architecting for a given microarchitecture. The requirements of a large base of contributors is part of the reason why RISC-V was engineered to address many possible uses.

The designers' primary assertion is that the instruction set is the key interface in a computer as it is situated at the interface between the hardware and the software. If a good instruction set were open and available for use by all, then it can dramatically reduce the cost of software by enabling far more reuse. It should also trigger increased competition among hardware providers, who can then devote more resources toward design and less for software support.

The designers maintain that new principles are becoming rare in instruction set design, as the most successful designs of the last forty years have grown increasingly similar. Of those that failed, most did so because their sponsoring companies were financially unsuccessful, not because the instruction sets were technically poor. Thus, a well-designed open instruction set designed using well-established principles should attract long-term support by many vendors.

RISC-V also encourages academic usage. The simplicity of the integer subset permits basic student exercises, and is a simple enough ISA to enable software to control

research machines. The variable-length ISA provides room for instruction set extensions for both student exercises and research, and the separated privileged instruction set permits research in operating system support without redesigning compilers. RISC-V's open intellectual property paradigm allows derivative designs to be published, reused, and modified.

Chapter 3
QEMU : Quick Emulator

3.1 What is QEMU?

QEMU, which stands for Quick Emulator, is a free and open-source software that allows you to run various operating systems and user-space programs on different hardware architectures. It can function as both an emulator and a virtual machine manager.

3.2 How does QEMU work?

QEMU works by emulating the hardware of a target system, such as a CPU, memory, and devices. This emulation allows you to run guest operating systems and programs that were designed for a different architecture on your host system. QEMU can achieve this emulation in two ways:

Dynamic binary translation (DBT): DBT translates the guest's instructions into instructions that the host CPU can understand and execute. This approach is slower than using hardware virtualization, but it does not require any special hardware support.

Kernel-based Virtual Machine (KVM): KVM is a Linux kernel module that allows QEMU to leverage hardware virtualization extensions on the host CPU. This can significantly improve the performance of emulation.
What can QEMU be used for?

3.3 Application of QEMU

Running legacy operating systems: You can use QEMU to run old operating systems that are no longer supported on modern hardware. This can be useful for testing software compatibility or running old games.

Developing and testing software for different architectures: QEMU can be used to develop and test software for embedded systems, mobile devices, and other platforms.

Creating virtual machines: QEMU can be used to create virtual machines for a variety of purposes, such as testing, development, and deployment.

Emulating user-space programs: QEMU can be used to emulate user-space programs for different architectures. This can be useful for running programs that are not available for your host system.

3.4 Benefits of using QEMU

Free and open-source: QEMU is free to use and modify, which makes it a popular choice for both personal and commercial use.

Cross-platform: QEMU is available for a wide range of platforms, including Linux, Windows, macOS, and more.

Versatile: QEMU can be used for a wide variety of tasks, making it a valuable tool for many users.

High performance: QEMU can achieve near-native performance when using KVM.

Chapter 4
RISC-V Boot Flow: A Multi-Stage Journey

Booting a RISC-V system involves a series of steps executed by different stages of software, each building upon the previous one. This multi-stage approach allows for flexibility and customization based on the specific needs of the platform. Here's a breakdown of the typical RISC-V boot flow:

4.1 Stage 1: Power On and Initialization

➔ **Power On Self Test (POST):** This triggers the processor to boot up.

➔ **Firmware Initialisation:** The firmware, stored in non-volatile memory like ROM, initializes basic hardware components like clocks and memory controllers.

➔ **First Stage Bootloader (FSBL):** The firmware hands over control to the FSBL, often residing in ROM or flash. This minimal program further initializes hardware and sets up the execution environment.

4.2 Stage 2: Bootloader & Runtime Services

➔ **Secondary Program Loader (SPL):** The FSBL loads and executes the SPL, which resides in external storage like flash memory.

➔ **Runtime Service Provider (RTSP):** The SPL initializes runtime services like memory management and communication interfaces, enabling access to peripheral devices.

➔ **Bootloader:** The RTSP loads and executes the main bootloader, which can be:

- **u-boot:** Popular for embedded systems, offering rich features like device drivers and command line interface.

- **OpenSBI (Open Source Base Interface):** A lightweight, vendor-neutral interface for interacting with platform services, often used in conjunction with u-boot or coreboot.

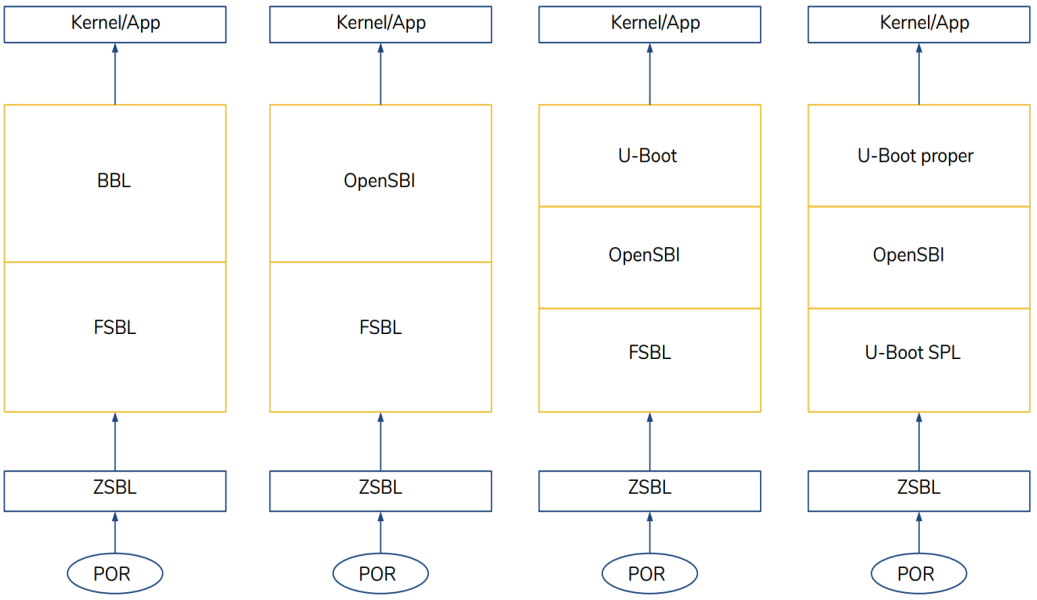
- **EDKII (EFI Development Kit II):** Implements the UEFI standard, allowing for secure boot and compatibility with existing bootloaders like GRUB.

4.3 Stage 3: Kernel and Operating System

➔ **Kernel Loading:** The chosen bootloader loads the operating system kernel (e.g., Linux) into memory.

➔ **Kernel Decompression and Setup:** The kernel decompresses itself, initializes devices, and mounts the root filesystem.

➔ **Operating System Startup:** The kernel launches the initial user processes, and the operating system starts running.



ZSBL: Zero Stage Bootloader(BROM), **BBL**: Berkeley Bootloader, **FSBL**: First Stage Bootloader, **OpenSBI**: RISC-V Open Source Supervisory Binary Interface

Fig. 1: RISC-V Boot Flow

Chapter 5
OpenSBI : Open Source Supervisory Binary Interface

In the RISC-V boot flow, OpenSBI (Open Source Supervisory Binary Interface) plays a crucial role as a vendor-neutral and lightweight interface between software running in S-mode (supervisor mode) and the underlying platform hardware. It provides essential services without dictating specific implementations, offering flexibility and customization for different RISC-V platforms.

5.1 Key Functionalities

- **Hardware Access:** OpenSBI offers an API for S-mode software to access and manage hardware resources like timers, interrupts, memory, and peripherals. This abstraction layer shields higher-level software from platform-specific hardware details.

- **Platform Services:** It provides basic platform services like console output, clock management, and power management, independent of the specific hardware implementation.

- **Bootloader Flexibility:** By separating hardware access from boot logic, OpenSBI enables various bootloaders like u-boot or coreboot to work with diverse RISC-V platforms.

5.2 Workflow

1. **Bootloader Activation:** During the boot process, a bootloader like u-boot initiates contact with OpenSBI by making specific service calls.

2. **Service Requests:** The bootloader uses OpenSBI's API to access hardware resources or request platform services. For example, it might call OpenSBI to configure memory, print messages to the console, or initiate network communication.

3. **Platform Interaction:** OpenSBI translates the bootloader's requests into platform-specific instructions and interacts with the underlying hardware accordingly. It utilizes platform-specific drivers and firmware to accomplish the tasks.

4. **Hardware Response:** OpenSBI relays the response from the hardware back to the bootloader, enabling it to proceed with the next steps in the boot process.

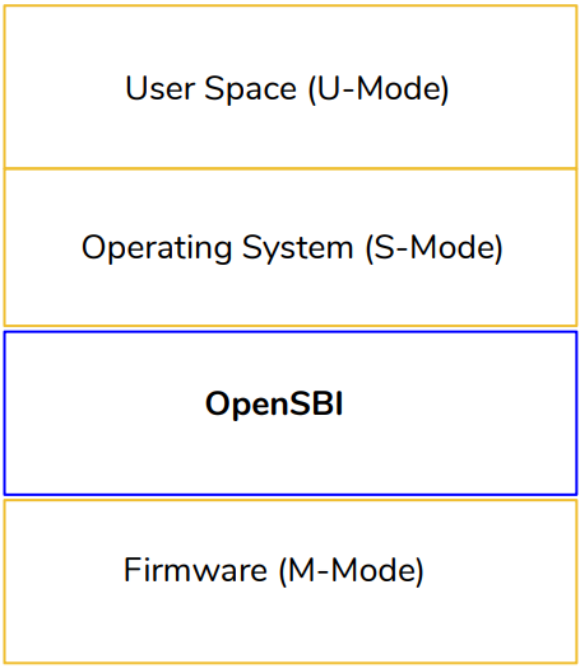


Fig. 2: OpenSBI in RISC-V Boot Flow

5.3 Benefits of OpenSBI

- **Abstraction and Portability:** OpenSBI's abstraction layer enables S-mode software to be more portable across different RISC-V platforms, reducing development complexity and effort.

- **Vendor Neutrality:** By avoiding vendor-specific implementations, OpenSBI fosters an open and competitive ecosystem for RISC-V software development.

- **Modular Design:** Its modular design allows for customization and extension based on specific platform needs.

OpenSBI's role is crucial in the RISC-V ecosystem, providing a standardised interface for diverse hardware platforms and enabling flexible boot solutions.

Chapter 6
Methodology- Bootloaders Used

In this project we have used three bootloaders namely: u-boot, edk2 and coreboot that are supported on RISC-V architecture. In this chapter the detailed description about these bootloaders is given which can be helpful for understanding the booting process for RISC-V machines.

6.1 u-boot

6.1.1 Introduction to u-boot

U-Boot can run in either M-mode or S-mode, depending on whether it runs before the initialization of the firmware providing SBI (Supervisor Binary Interface). The firmware is necessary in the RISC-V boot process as it serves as a SEE (Supervisor Execution Environment) to handle exceptions for the S-mode U-Boot or Operating System.

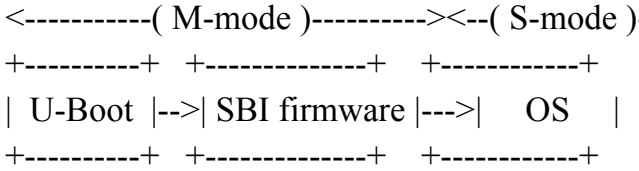
In between the boot phases, the hartid is passed through the a0 register, and the start address of the devicetree is passed through the a1 register.

As a reference, OpenSBI is an SBI implementation that can be used with U-Boot in different modes.

6.1.2 M-mode U-Boot

When running in M-mode U-Boot, it will load the payload image (e.g. fw_payload) which contains the firmware and the S-mode Operating System; in this case, you can use mkimage to package the payload image into an uImage format, and boot it using the bootm command.

The following diagram illustrates the boot process:

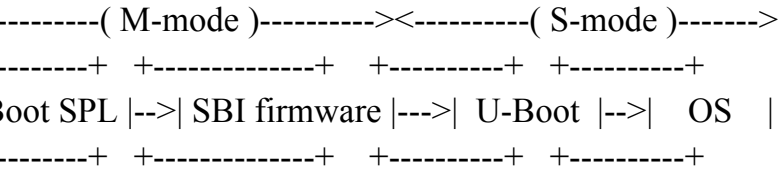


To examine the boot process with the QEMU virtual machine, the details are given in Chapter 7 Implementation of this document.

6.1.3 S-mode U-Boot

RISC-V production boot images may include a U-Boot SPL for platform-specific initialization. The U-Boot SPL then loads a FIT image (u-boot.itb), which contains a firmware (e.g. fw_dynamic) providing the SBI, as well as a regular U-Boot (or U-Boot proper) running in S-mode. Finally, the S-mode Operating System is loaded.

The following diagram illustrates the boot process:



6.2 EDK2

6.2.1 Introduction to EDK2

UEFI is an interface standard independent of the processor platform. It was originally designed to replace the old and slow traditional BIOS, promoting the innovation and development of computer systems. Since then, it has gradually become the mainstream of the market in different application scenarios such as PC and embedded systems.

The RISC-V instruction set, as a reduced instruction set, is designed with small size, fast speed and low power consumption to meet the needs . It is open source, modular and the tool chain is simple, and is therefore able to adapt to different application scenarios. At present, under the wave of the domestic chip market development, RISC-V chips are extended from the low-power embedded systems to the high-performance servers, which urgently needs UEFI boot support.

In April 2021, Western Digital developers provided a series of patches for Linux as some preparatory fixes for Linux's RISC-V UEFI support. Sifive has also made a lot of adaptations for its Sifive-U540 RISC-V processors. These all demonstrate that UEFI-based RISC-V boot firmware has great development value. On the other hand, the implementation of RISC-V UEFI is also capable of providing a boot solution independent of the operating systems. Previously, most RISC-V boot schemes including U-Boot and CoreBoot are closely related to specific operating systems.

However, the current RISC-V UEFI boot emulation based on the EDK2 implementation still has three defects: it only supports the sifive-U540 platform; it is not capable of modifying RISC-V hardware in QEMU; it does not support using the QEMU command to directly boot the operating system. Based on the three defects

illustrated above, this paper studies the principles and methods of RISC-V UEFI porting and optimization in the QEMU virtual environment.

6.2.2 RISC-V UEFI Boot Stages

UEFI boot stages can be roughly divided into SEC, PEI, DXE, BDS, TSL and RT. Among them, the SEC stage receives and processes the system startup and restart signals, initializes the temporary memory; serves as the system's root of trust; passes the system configuration to the next stage; the PEI stage mainly completes the initialization of the main memory, and passes the information required by the DXE stage through HOB list. The DXE stage realizes the platform initialization and provides boot-time services and run-time services for the operating system; the BDS stage executes the boot strategy and initialize system loader from the boot device.

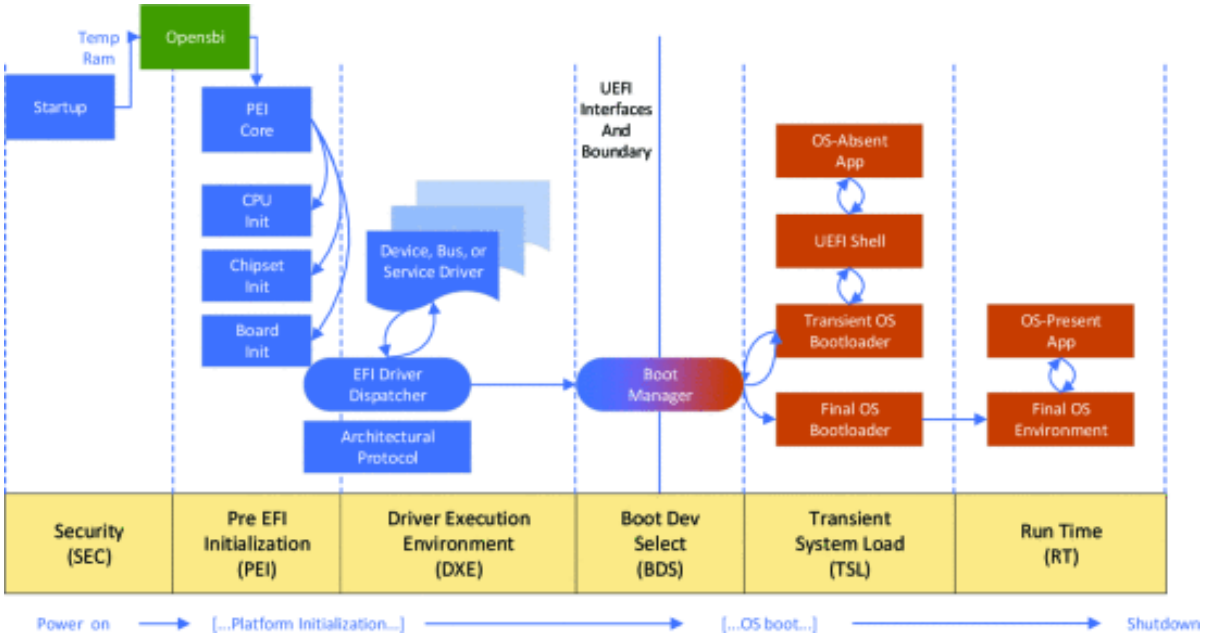


Figure 1.

6.2.3 RISC-V UEFI boot process

Opensbi is currently the most common implementation of the OS's binary interface for RISC-V, and has replaced the traditional BBL (Berkeley Boot Loader) as the first choice for RISC-V processor initialization. Nowadays, to boot the UEFI firmware on the RISC-V processor, the Opensbi + UEFI method is used . Here Opensbi is called by UEFI firmware as a library, and the boot stages . Specifically, in the first stage of UEFI after the system is powered on, sbi_init() is invoked by the SEC module to enter the OpenSBI library to complete the initialization of the hardware, including CPU and memory controller. After the execution is completed, the system is returned to the PEI stage of UEFI, and executes the subsequent initialization tasks. Since the memory

initialization has been completed by Opensbi, UEFI no longer implements most of the initialization tasks in SEC and PEI. It only needs to implement the processing of the start signal in SEC, the initialization of the temporary memory, and the establishment of the HOB lists in PEI. Besides, the DXE and BDS stages in this solution are the same as in the standard UEFI boot process.

6.3 Coreboot

Coreboot, formerly known as **LinuxBIOS** is a software project aimed at replacing proprietary firmware (BIOS or UEFI) found in most computers with a lightweight firmware designed to perform only the minimum number of tasks necessary to load and run a modern 32-bit or 64-bit operating system.

Since coreboot initializes the bare hardware, it must be ported to every chipset and motherboard that it supports. As a result, coreboot is available only for a limited number of hardware platforms and motherboard models.

6.3.1 Introduction to Coreboot

The coreboot project began with the goal of creating a BIOS that would start fast and handle errors intelligently.It is licensed under the terms of the **GNU General Public Licence version 2** (GPLv2).

CPU architectures supported by coreboot include x86_64, ARM, ARM64, MIPS and RISC-V.coreboot typically loads a Linux Kernel, but it can load any other ELF Stand-Alone executable that can boot a Linux kernel over a network, or SeaBIOS that can load a Linux kernel. Booting other kernels directly is also possible. Instead of loading a kernel directly, coreboot can pass control to a dedicated boot loader, such as a coreboot-capable version of GNU GRUB 2.

Coreboot is written primarily in C, with a small amount of assembly code. Choosing C as the primary programming language enables easier code audit when compared to contemporary PC BIOS that was generally written in assembly,which results in improved security. There is build and runtime support to write parts of coreboot in Ada to further raise the security bar, but it is currently only sporadically used. The source code is released under the GNU GPL Version 2 licence.

Coreboot performs the absolute minimal amount of hardware initialization and then passes control to the operating system. As a result, there is no coreboot code running once the operating system has taken control. A feature of coreboot is that the x86

version runs in 32-bit mode after executing only ten instructions (almost all other x86 BIOSes run exclusively in 16-bit mode). This is similar to the modern UEFI firmware, which is used on newer PC hardware.

The most difficult hardware that coreboot initializes is the DRAM Controllers and DRAM. RAM initialization is particularly difficult because before the RAM is initialized it cannot be used. Therefore, to initialize DRAM controllers and DRAM, the initialization code may have only the CPU's general purpose registers or Cache-as-RAM as temporary storage.

6.3.2 Advantages of Coreboot

1. Unlike standard BIOS, coreboot supports many features not found in traditional PC BIOS's such as automated testing, fast boot times, and low energy consumption. Core Boot also allows for greater control over the system's resources because it runs before any operating system is loaded onto the computer.

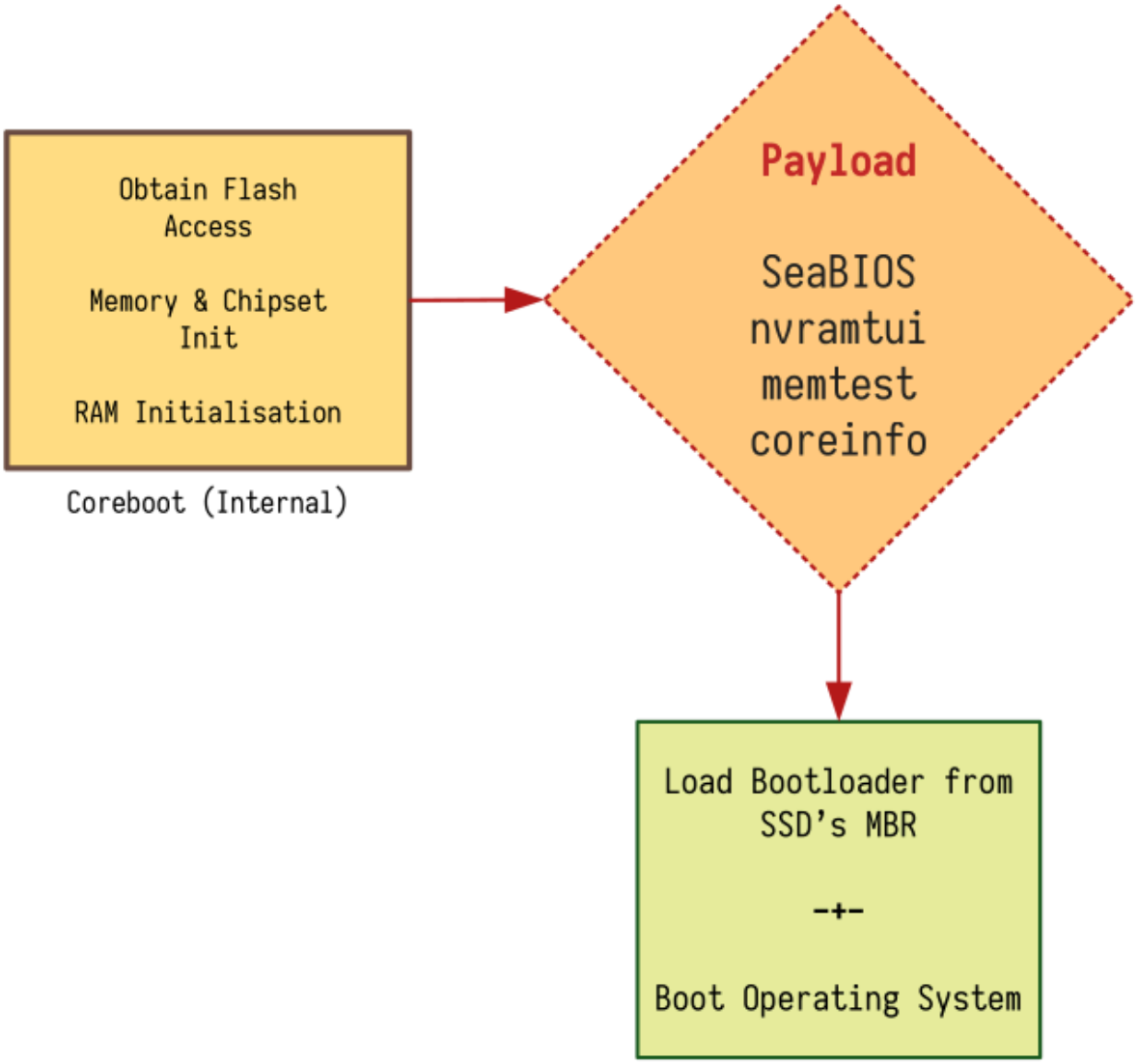
2. Coreboot is a project to develop open source boot firmware for various architectures. Its design philosophy is to do the bare minimum necessary to ensure that hardware is usable and then pass control to a different program called the payload.

3. Coreboot is an open-source project, which means the source code is publicly available and can be audited by security researchers and developers. This transparency helps identify and fix vulnerabilities, making coreboot potentially more secure than closed-source firmware.

4. Due to how lightweight coreboot is, it will offer better performance and lower power consumption.

6.2.3 Booting Stages

Coreboot consists of multiple stages that are compiled as separate binaries and are inserted into the CBFS with custom compression. The bootblock usually doesn’t have compression while the ramstage and payload are compressed with LZMA.Each stage loads the next stage at given address (possibly decompressing it).Some stages are relocatable and can be placed anywhere in DRAM. Those stages are usually cached in CBMEM for faster loading times on ACPI S3 resume.



Bootblock

The bootblock is the first stage executed after CPU reset. It is written in assembly language and its main task is to set up everything for a C-environment. Common task includes

- Cache-As-RAM for heap and stack
- Set stack pointer
- Clear memory for BSS
- Decompress and load the next stage.

Cache-as-RAM

The *Cache-As-Ram*, also called Non-Eviction mode, or *CAR, allows* to use the CPU cache like regular SRAM. This is particularly useful for high level languages like C,

which need RAM for heap and stack.The following stages run when Cache-As-Ram is active:

- bootblock
- romstage
- verstage
- postcar

Verstage

The verstage is where the root-of-trust starts. It's assumed that it cannot be overwritten in-field (together with the public key) and it starts at the very beginning of the boot process. The verstage installs a hook to verify a file before it's loaded from CBFS or a partition before it's accessed.
The verified boot mechanism allows trusted in-field firmware updates combined with a fail-safe recovery mode.

ROM stage

The romstage initializes the DRAM and prepares everything for device init.

Common tasks:

- Early device init
- DRAM init

Postcar

To leave the CAR setup and run code from regular DRAM the postcar-stage tears down CAR and loads the ramstage. Compared to other stages it's minimal in size.

RAM stage

The ramstage does the main device init:

- PCI device init
- On-chip device init
- TPM init (if not done by verstage)
- Graphics init (optional)
- CPU init (like set up SMM)

After initialization tables are written to inform the payload or operating system about the current hardware existence and state. That includes:

- ACPI tables (x86 specific)

Payloads

Coreboot doesn't try to mandate how the boot process should look, it merely does hardware init and then passes on control to another piece of software that we carry along in firmware storage, the *payload*.

1. **SeaBIOS:**SeaBIOS is an open source implementation of the PCBIOS API that exists since the original IBM PC and was extended since. While originally written for emulators such as QEMU, it can be built as a coreboot payload. It supports executing Option ROMs in a more complete fashion than coreboot.
2. **EDK2:**EDK2 is an open-source modern, feature-rich, cross-platform firmware development environment for the UEFI and UEFI Platform Initialization (PI) specifications.
3. **GRUB2:**GRUB2 was originally written as a bootloader and that's its most popular purpose, but it can also be compiled as a coreboot payload.
4. **LINUX:**There are several projects using Linux as a payload (which was the configuration that gave coreboot its original name, LinuxBIOS). That kernel is often rather small and serves to load a current kernel from somewhere, e.g. disk or network**.**

Heads

Heads is a distribution that bundles coreboot, Linux, busybox and custom tools to provide reproducible ROMs. Heads is an open source custom firmware and OS configuration for laptops and servers that aims to provide slightly better physical security and protection for data on the system.

Chapter 7
Implementation

The bootloaders mentioned in chapter 6 are implemented in the QEMU emulator. In this chapter we have mentioned the commands used to run the bootloaders also the screenshots of each step are given for better understanding for the readers.

7.1 u-boot implementation in QEMU

Following steps/commands are used to implement u-boot in the QEMU emulator.

7.1.1 Building U-Boot

Set the CROSS_COMPILE environment variable as usual, and run:

For 32-bit RISC-V:

```
$ make qemu-riscv32_defconfig
$ Make
```

For 64-bit RISC-V:

```
$ make qemu-riscv64_defconfig
$ make
```

This will compile U-Boot for machine mode. To build supervisor mode binaries, use the configurations qemu-riscv32_smode_defconfig and qemu-riscv64_smode_defconfig instead. Note that U-Boot running in supervisor mode requires a supervisor binary interface (SBI), such as RISC-V OpenSBI.

7.1.2 Running U-Boot

The minimal QEMU command line to get U-Boot up and running is:

For 64-bit RISC-V virt machine:

```
$ qemu-system-riscv64 -nographic -machine virt -bios u-boot.bin
```

The commands above create targets with 128MiB memory by default. A freely configurable amount of RAM can be created via the ‘-m’ parameter. For example, ‘-m

2G' creates 2GiB memory for the target, and the memory node in the embedded DTB created by QEMU reflects the new setting.

7.1.3 Running U-Boot SPL

In the default SPL configuration, U-Boot SPL starts in machine mode. U-Boot proper and OpenSBI (FW_DYNAMIC firmware) are bundled as FIT image and made available to U-Boot SPL. Both are then loaded by U-Boot SPL and the location of U-Boot proper is passed to OpenSBI. After initialization, U-Boot proper is started in supervisor mode by OpenSBI.

OpenSBI must be compiled before compiling U-Boot. Version 0.4 and higher is supported by U-Boot.

Clone the OpenSBI repository and run the following command:

```
$ git clone https://github.com/riscv/opensbi.git
$ cd opensbi
$ make PLATFORM=generic
```

To make the FW_DYNAMIC binary (build/platform/generic/firmware/fw_dynamic.bin) available to U-Boot, either copy it into the U-Boot root directory or specify its location with the OPENSBI environment variable. Afterwards, compile U-Boot with the following commands:

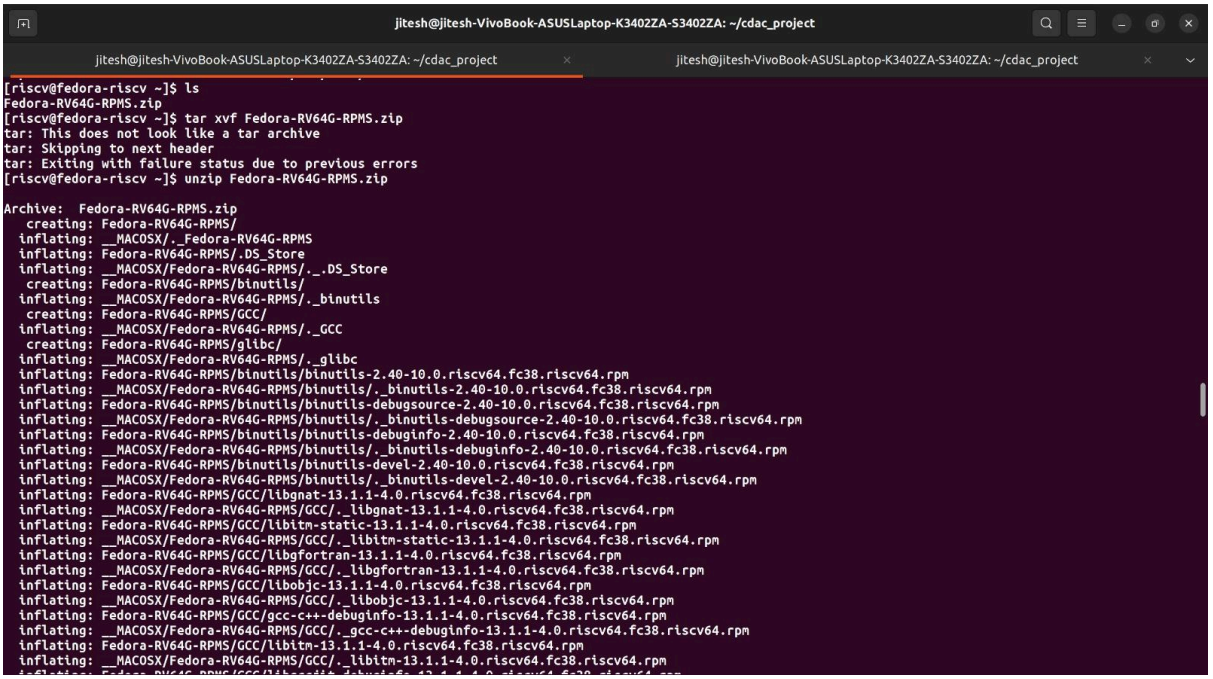
For 64-bit RISC-V:

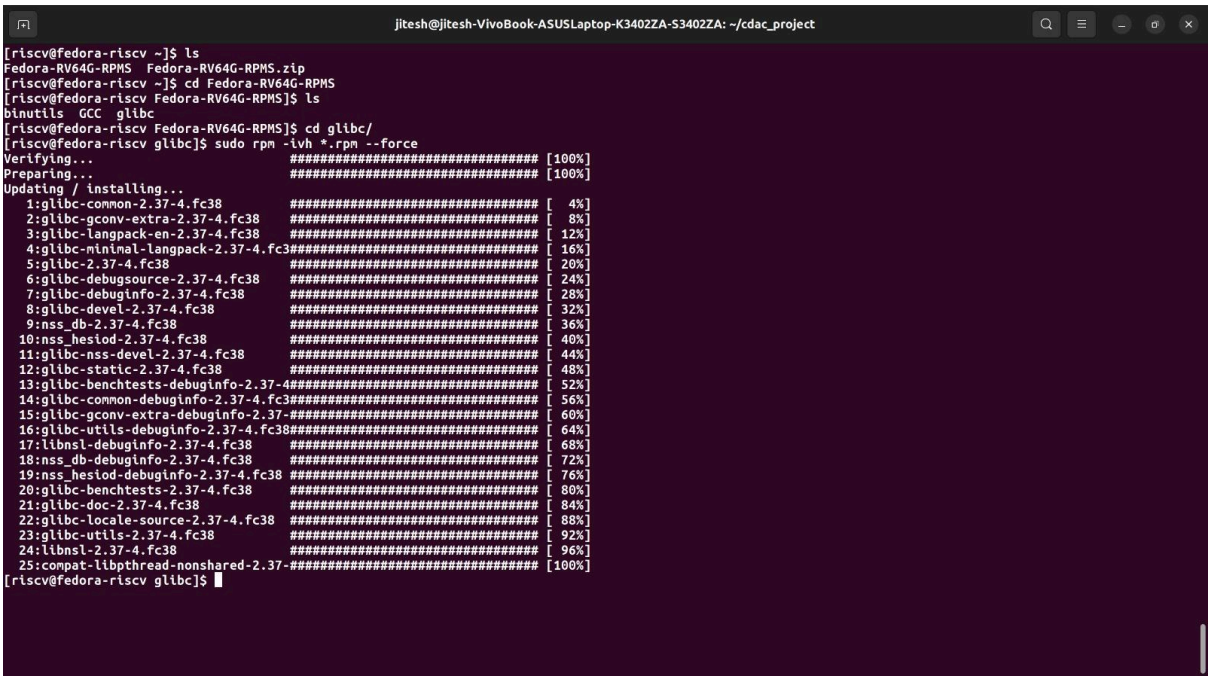
```
$ make qemu-riscv64_spl_defconfig
$ make
```

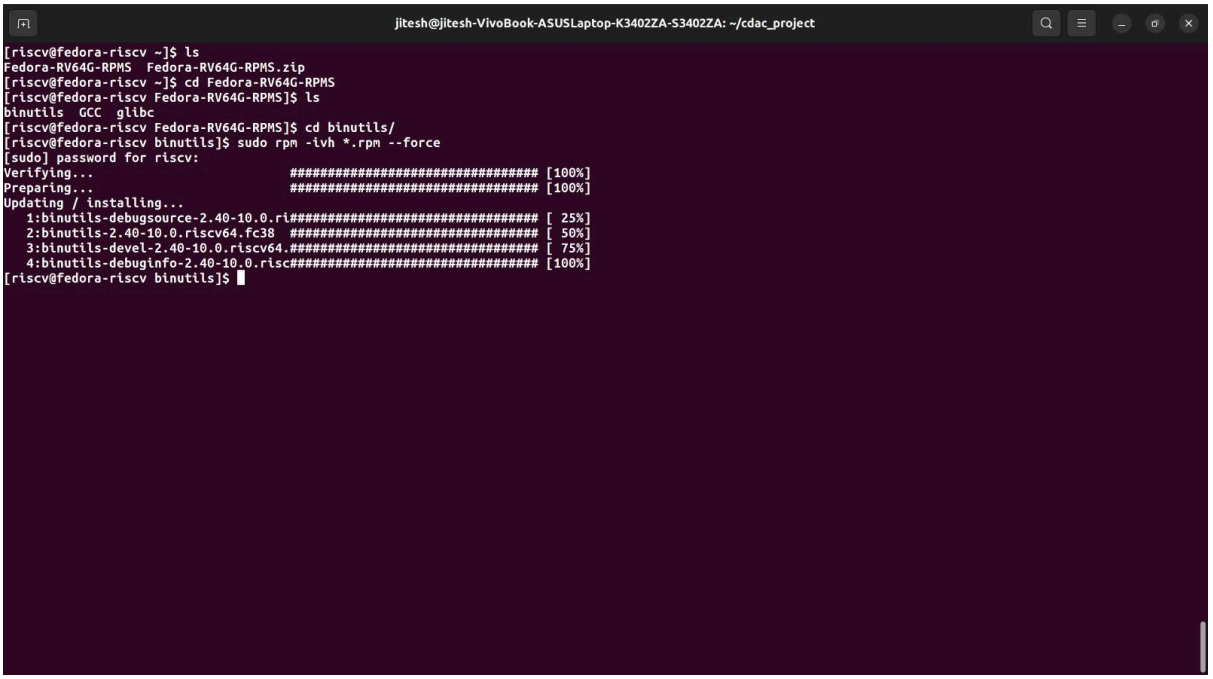
The minimal QEMU commands to run U-Boot SPL in 64-bit configurations are:

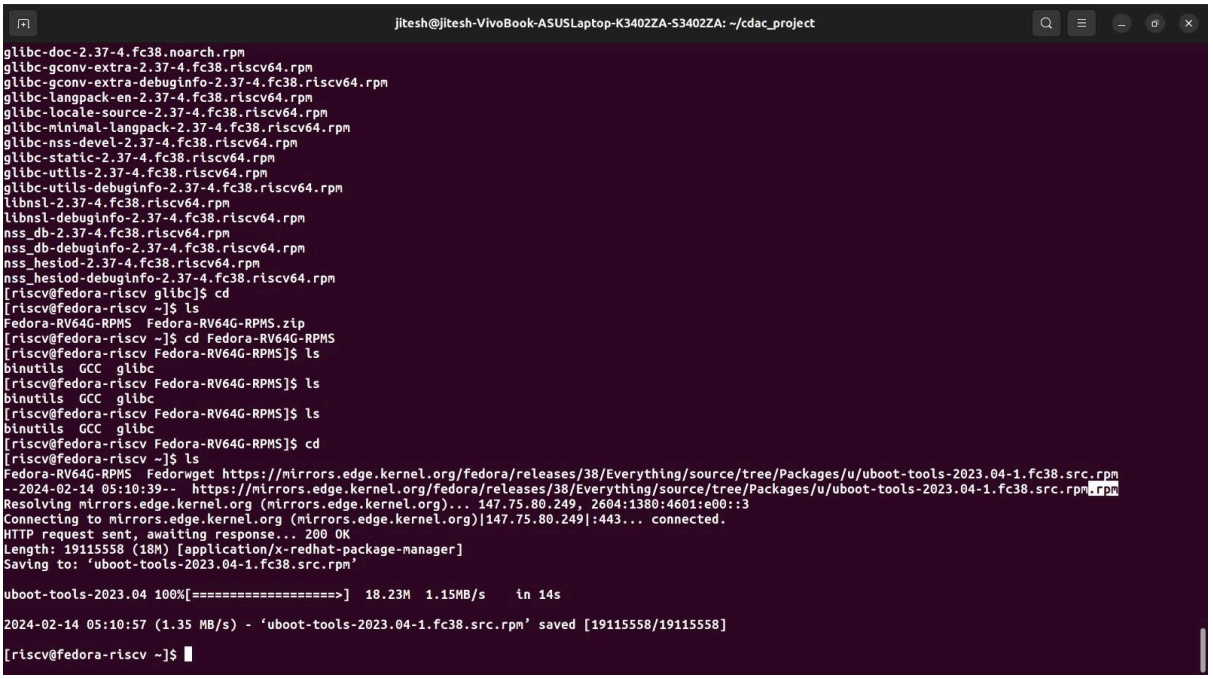
For 64-bit RISC-V virt machine:

```
$ qemu-system-riscv64 -nographic -machine virt -bios spl/u-boot-spl.bin -device loader,file=u-boot.itb,addr=0x80200000
```


jitesh@jitesh-VivoBook-ASUSLaptop-K3402ZA-S3402ZA: ~/cdac_project
jitesh@jitesh-VivoBook-ASUSLaptop-K3402ZA-S3402ZA: ~/cdac_project
jitesh@jitesh-VivoBook-ASUSLaptop-K3402ZA-S3402ZA: ~/cdac_project
[riscv@fedora-riscv ~]$ ls
Fedora-RV64G-RPMS.zip
[riscv@fedora-riscv ~]$ tar xvf Fedora-RV64G-RPMS.zip
tar: This does not look like a tar archive
tar: Skipping to next header
tar: Exiting with failure status due to previous errors
[riscv@fedora-riscv ~]$ unzip Fedora-RV64G-RPMS.zip
Archive: Fedora-RV64G-RPMS.zip
creating: Fedora-RV64G-RPMS/
inflating: __MACOSX/._Fedora-RV64G-RPMS
inflating: Fedora-RV64G-RPMS/.DS_Store
inflating: __MACOSX/Fedora-RV64G-RPMS/._.DS_Store
creating: Fedora-RV64G-RPMS/binutils/
inflating: __MACOSX/Fedora-RV64G-RPMS/._binutils
creating: Fedora-RV64G-RPMS/GCC/
inflating: __MACOSX/Fedora-RV64G-RPMS/._GCC
creating: Fedora-RV64G-RPMS/glibc/
inflating: __MACOSX/Fedora-RV64G-RPMS/._glibc
inflating: Fedora-RV64G-RPMS/binutils/binutils-2.40-10.0.riscv64.fc38.riscv64.rpm
inflating: __MACOSX/Fedora-RV64G-RPMS/binutils/._binutils-2.40-10.0.riscv64.fc38.riscv64.rpm
inflating: Fedora-RV64G-RPMS/binutils/binutils-debugsource-2.40-10.0.riscv64.fc38.riscv64.rpm
inflating: __MACOSX/Fedora-RV64G-RPMS/binutils/._binutils-debugsource-2.40-10.0.riscv64.fc38.riscv64.rpm
inflating: Fedora-RV64G-RPMS/binutils/binutils-debuginfo-2.40-10.0.riscv64.fc38.riscv64.rpm
inflating: __MACOSX/Fedora-RV64G-RPMS/binutils/._binutils-debuginfo-2.40-10.0.riscv64.fc38.riscv64.rpm
inflating: Fedora-RV64G-RPMS/binutils/binutils-devel-2.40-10.0.riscv64.fc38.riscv64.rpm
inflating: __MACOSX/Fedora-RV64G-RPMS/binutils/._binutils-devel-2.40-10.0.riscv64.fc38.riscv64.rpm
inflating: Fedora-RV64G-RPMS/GCC/libgnat-13.1.1-4.0.riscv64.fc38.riscv64.rpm
inflating: __MACOSX/Fedora-RV64G-RPMS/GCC/._libgnat-13.1.1-4.0.riscv64.fc38.riscv64.rpm
inflating: Fedora-RV64G-RPMS/GCC/libitm-static-13.1.1-4.0.riscv64.fc38.riscv64.rpm
inflating: __MACOSX/Fedora-RV64G-RPMS/GCC/._libitm-static-13.1.1-4.0.riscv64.fc38.riscv64.rpm
inflating: Fedora-RV64G-RPMS/GCC/libgfortran-13.1.1-4.0.riscv64.fc38.riscv64.rpm
inflating: __MACOSX/Fedora-RV64G-RPMS/GCC/._libgfortran-13.1.1-4.0.riscv64.fc38.riscv64.rpm
inflating: Fedora-RV64G-RPMS/GCC/libobjc-13.1.1-4.0.riscv64.fc38.riscv64.rpm
inflating: __MACOSX/Fedora-RV64G-RPMS/GCC/._libobjc-13.1.1-4.0.riscv64.fc38.riscv64.rpm
inflating: Fedora-RV64G-RPMS/GCC/gcc-c++-debuginfo-13.1.1-4.0.riscv64.fc38.riscv64.rpm
inflating: __MACOSX/Fedora-RV64G-RPMS/GCC/._gcc-c++-debuginfo-13.1.1-4.0.riscv64.fc38.riscv64.rpm
inflating: Fedora-RV64G-RPMS/GCC/libitm-13.1.1-4.0.riscv64.fc38.riscv64.rpm
inflating: __MACOSX/Fedora-RV64G-RPMS/GCC/._libitm-13.1.1-4.0.riscv64.fc38.riscv64.rpm


jitesh@jitesh-VivoBook-ASUSLaptop-K3402ZA-S3402ZA: ~/cdac_project
[riscv@fedora-riscv ~]$ ls
Fedora-RV64G-RPMS Fedora-RV64G-RPMS.zip
[riscv@fedora-riscv ~]$ cd Fedora-RV64G-RPMS
[riscv@fedora-riscv Fedora-RV64G-RPMS]$ ls
binutils GCC glibc
[riscv@fedora-riscv Fedora-RV64G-RPMS]$ cd glibc/
[riscv@fedora-riscv glibc]$ sudo rpm -ivh *.rpm --force
Verifying... ## [100%]
Preparing... ## [100%]
Updating / installing...
1:glibc-common-2.37-4.fc38 ## [4%]
2:glibc-gconv-extra-2.37-4.fc38 ## [8%]
3:glibc-langpack-en-2.37-4.fc38 ## [12%]
4:glibc-minimal-langpack-2.37-4.fc3## [16%]
5:glibc-2.37-4.fc38 ## [20%]
6:glibc-debugsource-2.37-4.fc38 ## [24%]
7:glibc-debuginfo-2.37-4.fc38 ## [28%]
8:glibc-devel-2.37-4.fc38 ## [32%]
9:nss_db-2.37-4.fc38 ## [36%]
10:nss_hesiod-2.37-4.fc38 ## [40%]
11:glibc-nss-devel-2.37-4.fc38 ## [44%]
12:glibc-static-2.37-4.fc38 ## [48%]
13:glibc-benchtests-debuginfo-2.37-4## [52%]
14:glibc-common-debuginfo-2.37-4.fc3## [56%]
15:glibc-gconv-extra-debuginfo-2.37-## [60%]
16:glibc-utils-debuginfo-2.37-4.fc38## [64%]
17:libnsl-debuginfo-2.37-4.fc38 ## [68%]
18:nss_db-debuginfo-2.37-4.fc38 ## [72%]
19:nss_hesiod-debuginfo-2.37-4.fc38 ## [76%]
20:glibc-benchtests-2.37-4.fc38 ## [80%]
21:glibc-doc-2.37-4.fc38 ## [84%]
22:glibc-locale-source-2.37-4.fc38 ## [88%]
23:glibc-utils-2.37-4.fc38 ## [92%]
24:libnsl-2.37-4.fc38 ## [96%]
25:compat-libpthread-nonshared-2.37-## [100%]
[riscv@fedora-riscv glibc]$

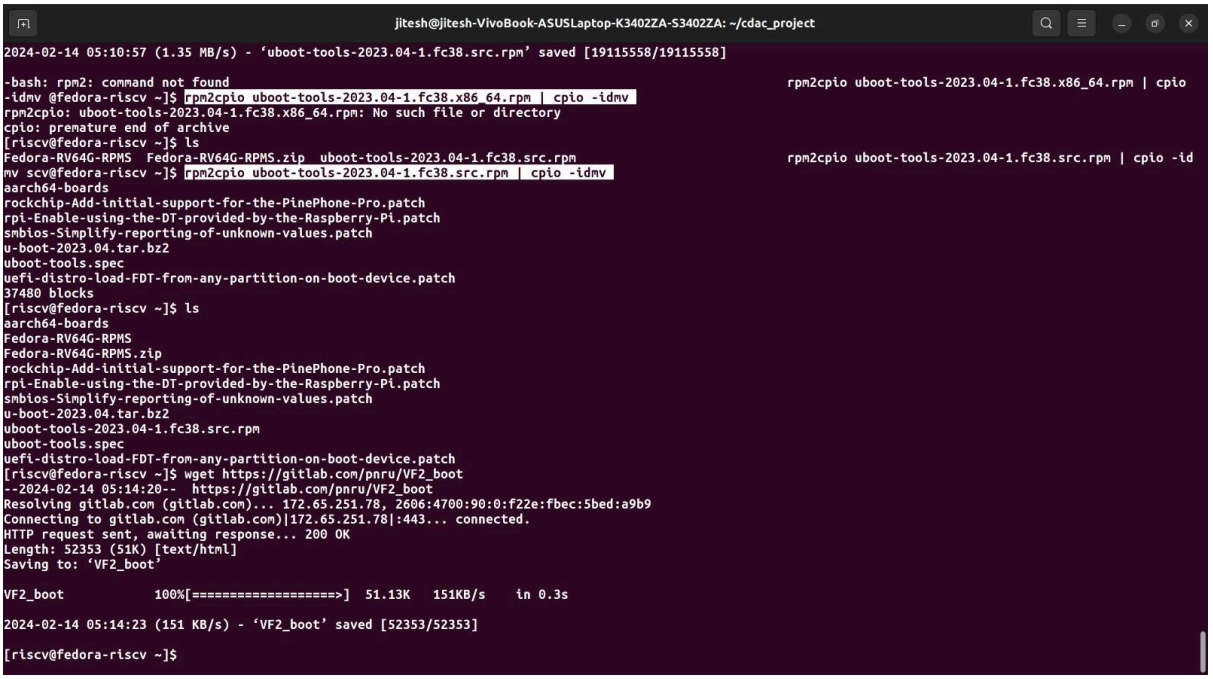

jitesh@jitesh-VivoBook-ASUSLaptop-K3402ZA-S3402ZA: ~/cdac_project
[riscv@fedora-riscv ~]$ ls
Fedora-RV64G-RPMS Fedora-RV64G-RPMS.zip
[riscv@fedora-riscv ~]$ cd Fedora-RV64G-RPMS
[riscv@fedora-riscv Fedora-RV64G-RPMS]$ ls
binutils GCC glibc
[riscv@fedora-riscv Fedora-RV64G-RPMS]$ cd binutils/
[riscv@fedora-riscv binutils]$ sudo rpm -ivh *.rpm --force
[sudo] password for riscv:
Verifying...   ################################# [100%]
Preparing...   ################################# [100%]
Updating / installing...
   1:binutils-debugsource-2.40-10.0.ri################################# [25%]
   2:binutils-2.40-10.0.riscv64.fc38 ################################# [50%]
   3:binutils-devel-2.40-10.0.riscv64.################################# [75%]
   4:binutils-debuginfo-2.40-10.0.risc################################# [100%]
[riscv@fedora-riscv binutils]$

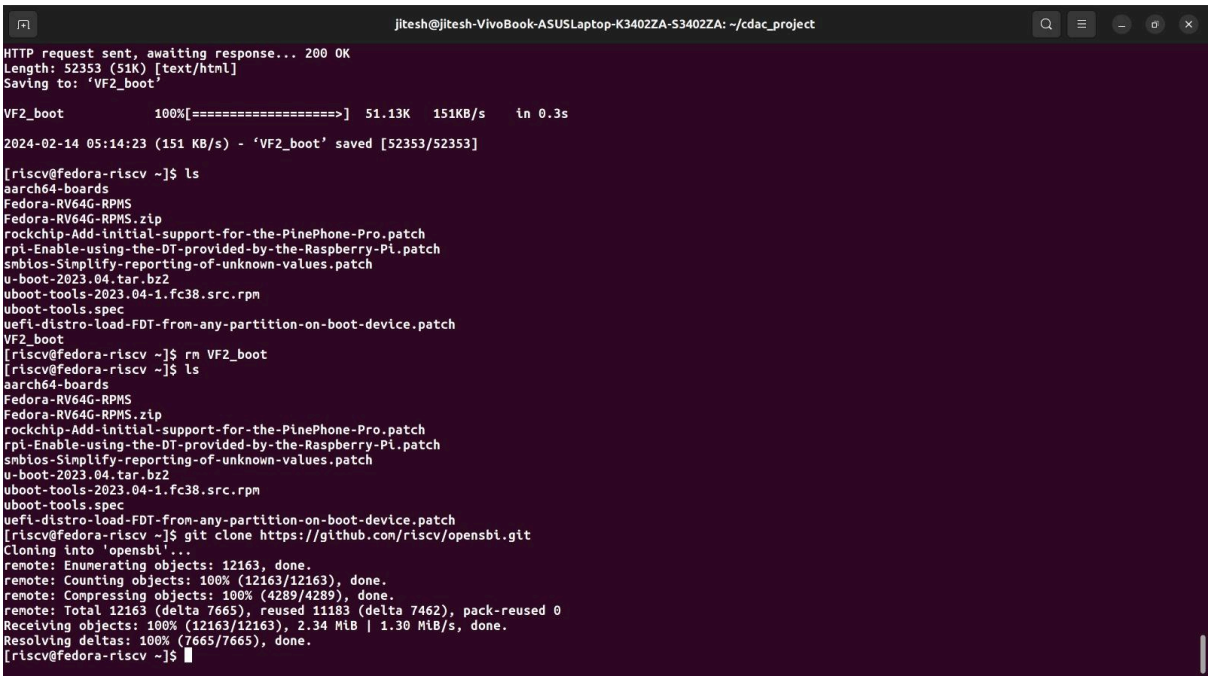

jitesh@jitesh-VivoBook-ASUSLaptop-K3402ZA-S3402ZA: ~/cdac_project
glibc-doc-2.37-4.fc38.noarch.rpm
glibc-gconv-extra-2.37-4.fc38.riscv64.rpm
glibc-gconv-extra-debuginfo-2.37-4.fc38.riscv64.rpm
glibc-langpack-en-2.37-4.fc38.riscv64.rpm
glibc-locale-source-2.37-4.fc38.riscv64.rpm
glibc-minimal-langpack-2.37-4.fc38.riscv64.rpm
glibc-nss-devel-2.37-4.fc38.riscv64.rpm
glibc-static-2.37-4.fc38.riscv64.rpm
glibc-utils-2.37-4.fc38.riscv64.rpm
glibc-utils-debuginfo-2.37-4.fc38.riscv64.rpm
libnsl-2.37-4.fc38.riscv64.rpm
libnsl-debuginfo-2.37-4.fc38.riscv64.rpm
nss_db-2.37-4.fc38.riscv64.rpm
nss_db-debuginfo-2.37-4.fc38.riscv64.rpm
nss_hesiod-2.37-4.fc38.riscv64.rpm
nss_hesiod-debuginfo-2.37-4.fc38.riscv64.rpm
[riscv@fedora-riscv glibc]$ cd
[riscv@fedora-riscv ~]$ ls
Fedora-RV64G-RPMS Fedora-RV64G-RPMS.zip
[riscv@fedora-riscv ~]$ cd Fedora-RV64G-RPMS
[riscv@fedora-riscv Fedora-RV64G-RPMS]$ ls
binutils GCC glibc
[riscv@fedora-riscv Fedora-RV64G-RPMS]$ ls
binutils GCC glibc
[riscv@fedora-riscv Fedora-RV64G-RPMS]$ ls
binutils GCC glibc
[riscv@fedora-riscv Fedora-RV64G-RPMS]$ cd
[riscv@fedora-riscv ~]$ ls
Fedora-RV64G-RPMS Fedorwget https://mirrors.edge.kernel.org/fedora/releases/38/Everything/source/tree/Packages/u/uboot-tools-2023.04-1.fc38.src.rpm
--2024-02-14 05:10:39-- https://mirrors.edge.kernel.org/fedora/releases/38/Everything/source/tree/Packages/u/uboot-tools-2023.04-1.fc38.src.rpm.rpm
Resolving mirrors.edge.kernel.org (mirrors.edge.kernel.org)... 147.75.80.249, 2604:1380:4601:e00::3
Connecting to mirrors.edge.kernel.org (mirrors.edge.kernel.org)|147.75.80.249|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 19115558 (18M) [application/x-redhat-package-manager]
Saving to: 'uboot-tools-2023.04-1.fc38.src.rpm'

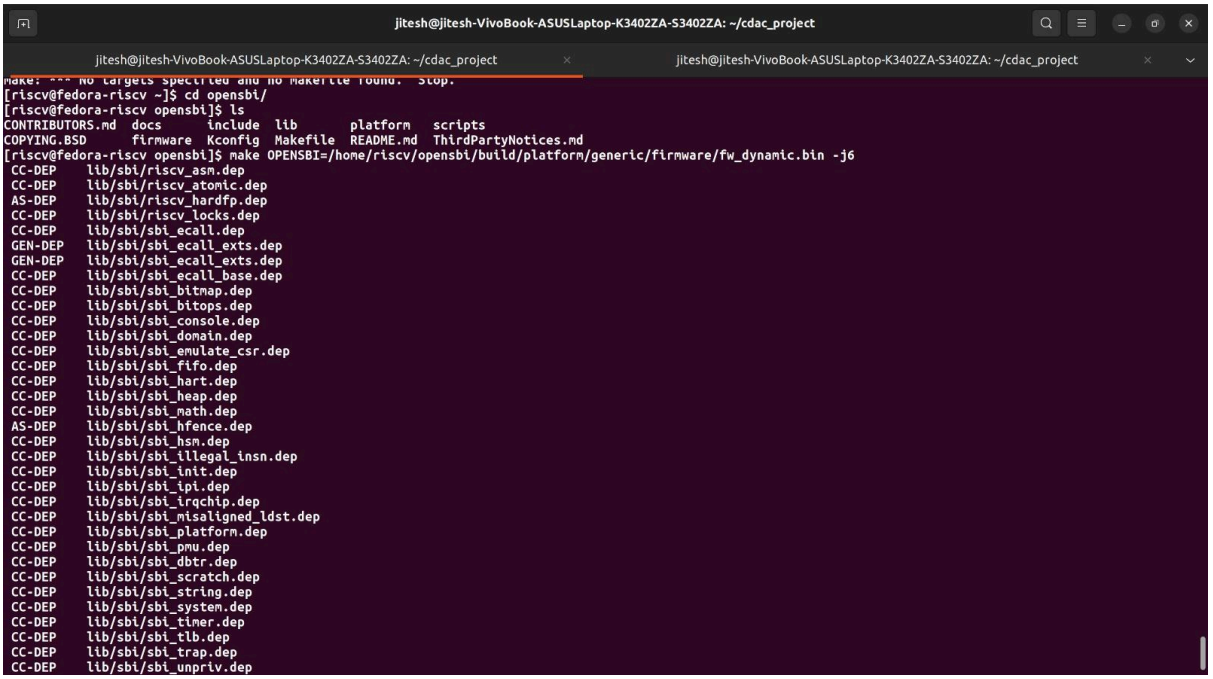
uboot-tools-2023.04 100%[==================>] 18.23M 1.15MB/s   in 14s

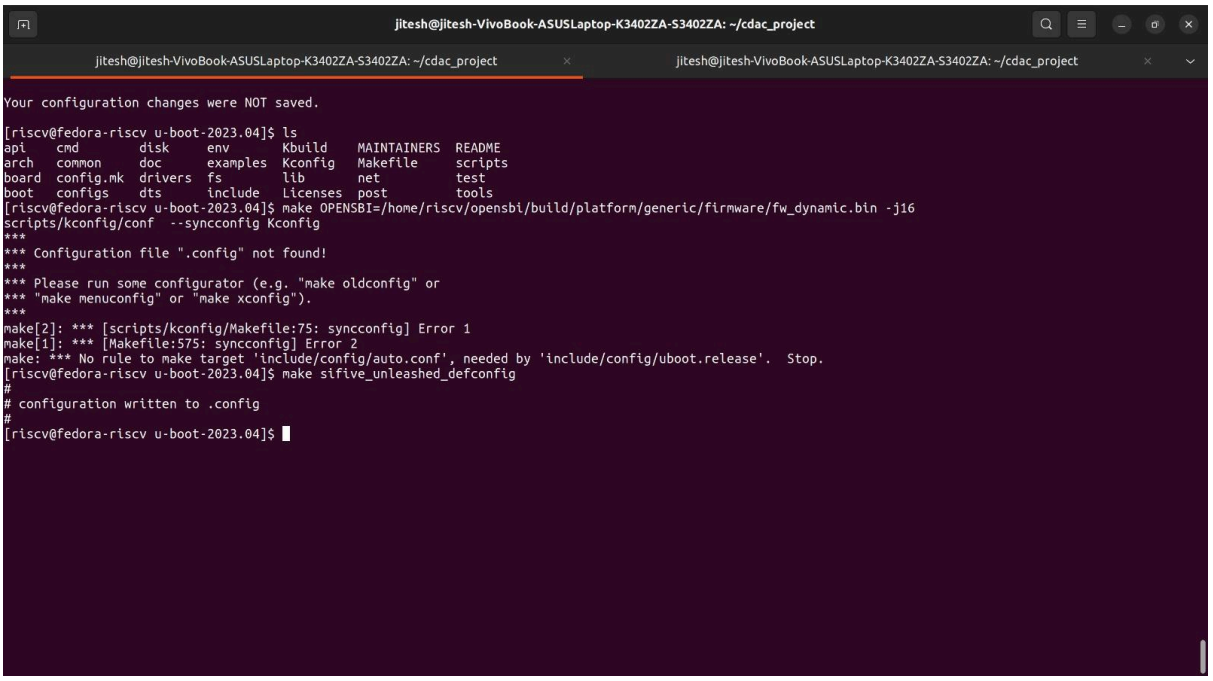
2024-02-14 05:10:57 (1.35 MB/s) - 'uboot-tools-2023.04-1.fc38.src.rpm' saved [19115558/19115558]

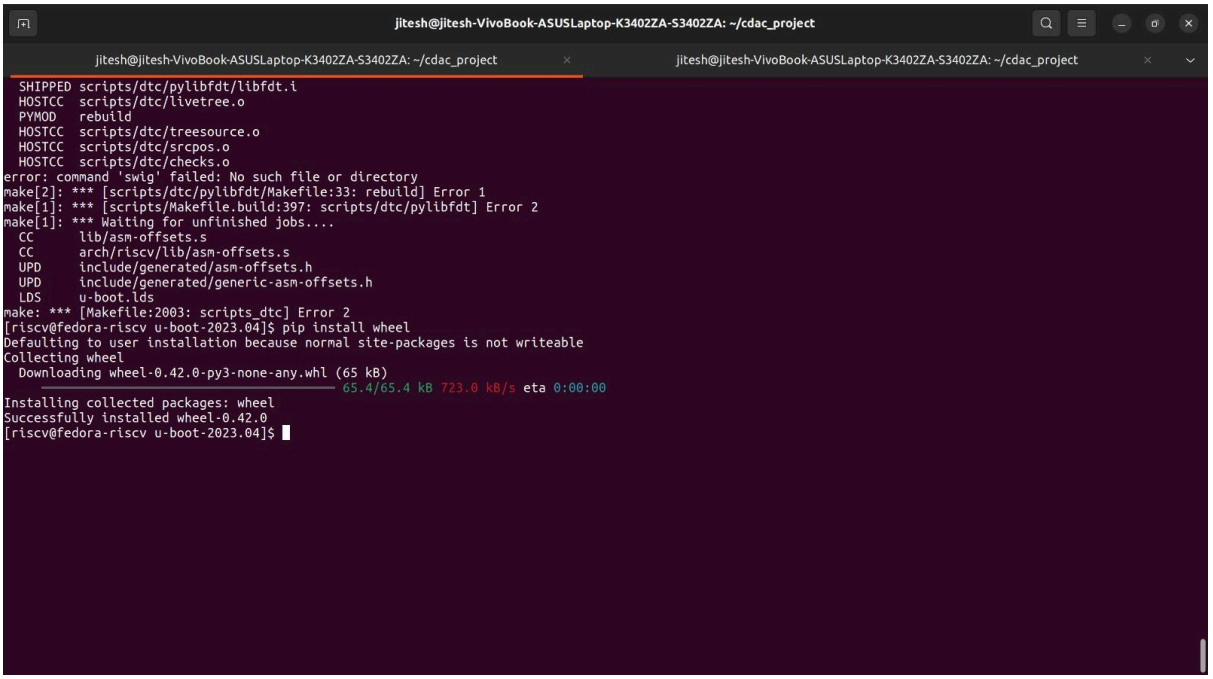
[riscv@fedora-riscv ~]$

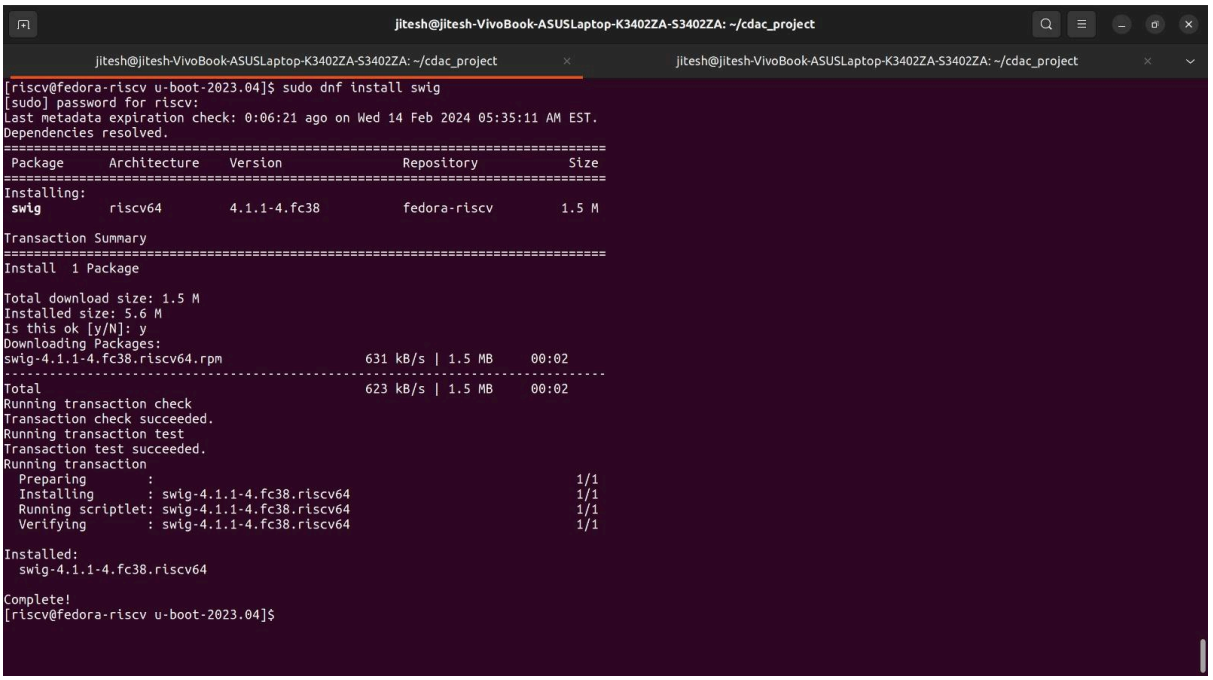

jitesh@jitesh-VivoBook-ASUSLaptop-K3402ZA-S3402ZA: ~/cdac_project
2024-02-14 05:10:57 (1.35 MB/s) - ‘uboot-tools-2023.04-1.fc38.src.rpm’ saved [19115558/19115558]
-bash: rpm2: command not found
rpm2cpio uboot-tools-2023.04-1.fc38.x86_64.rpm | cpio
-idmv @fedora-riscv ~]$ rpm2cpio uboot-tools-2023.04-1.fc38.x86_64.rpm | cpio -idmv
rpm2cpio: uboot-tools-2023.04-1.fc38.x86_64.rpm: No such file or directory
cpio: premature end of archive
[riscv@fedora-riscv ~]$ ls
Fedora-RV64G-RPMS Fedora-RV64G-RPMS.zip uboot-tools-2023.04-1.fc38.src.rpm
rpm2cpio uboot-tools-2023.04-1.fc38.src.rpm | cpio -id
mv scv@fedora-riscv ~]$ rpm2cpio uboot-tools-2023.04-1.fc38.src.rpm | cpio -idmv
aarch64-boards
rockchip-Add-initial-support-for-the-PinePhone-Pro.patch
rpi-Enable-using-the-DT-provided-by-the-Raspberry-Pi.patch
smbios-Simplify-reporting-of-unknown-values.patch
u-boot-2023.04.tar.bz2
uboot-tools.spec
uefi-distro-load-FDT-from-any-partition-on-boot-device.patch
37480 blocks
[riscv@fedora-riscv ~]$ ls
aarch64-boards
Fedora-RV64G-RPMS
Fedora-RV64G-RPMS.zip
rockchip-Add-initial-support-for-the-PinePhone-Pro.patch
rpi-Enable-using-the-DT-provided-by-the-Raspberry-Pi.patch
smbios-Simplify-reporting-of-unknown-values.patch
u-boot-2023.04.tar.bz2
uboot-tools-2023.04-1.fc38.src.rpm
uboot-tools.spec
uefi-distro-load-FDT-from-any-partition-on-boot-device.patch
[riscv@fedora-riscv ~]$ wget https://gitlab.com/pnru/VF2_boot
--2024-02-14 05:14:20-- https://gitlab.com/pnru/VF2_boot
Resolving gitlab.com (gitlab.com)... 172.65.251.78, 2606:4700:90:0:f22e:fbec:5bed:a9b9
Connecting to gitlab.com (gitlab.com)|172.65.251.78|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 52353 (51K) [text/html]
Saving to: ‘VF2_boot’
VF2_boot 100%[==========>] 51.13K 151KB/s in 0.3s
2024-02-14 05:14:23 (151 KB/s) - ‘VF2_boot’ saved [52353/52353]
[riscv@fedora-riscv ~]$

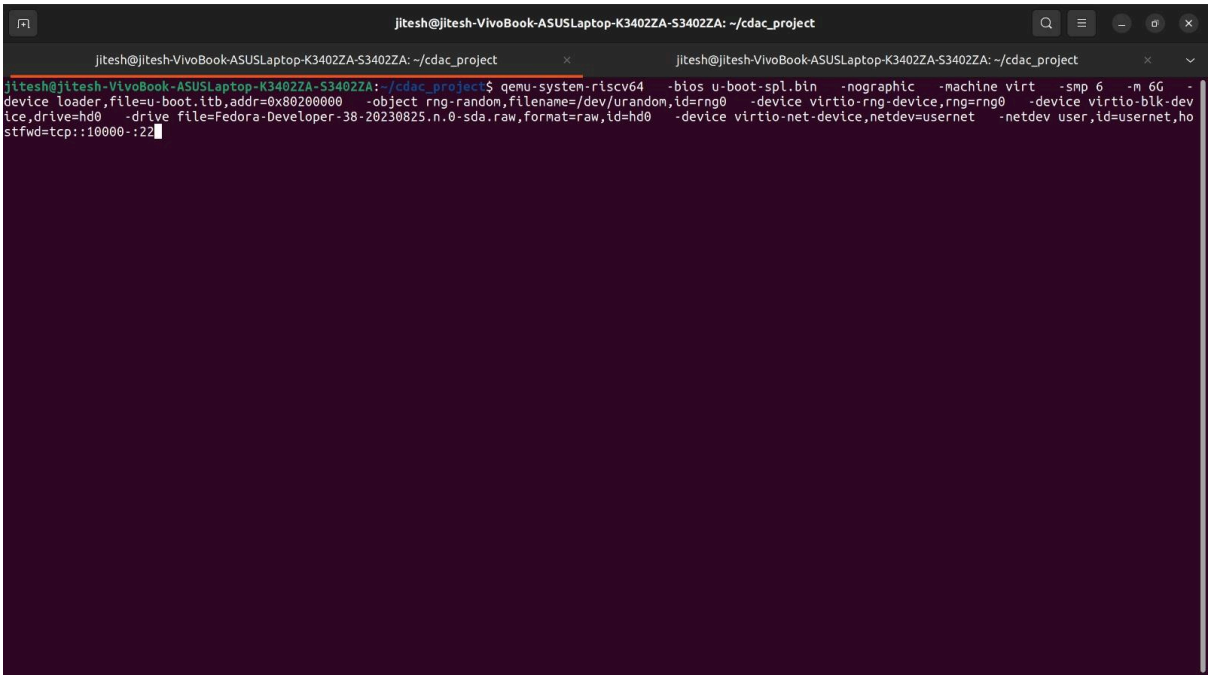

jitesh@jitesh-VivoBook-ASUSLaptop-K3402ZA-S3402ZA: ~/cdac_project
HTTP request sent, awaiting response... 200 OK
Length: 52353 (51K) [text/html]
Saving to: ‘VF2_boot’
VF2_boot 100%[==========>] 51.13K 151KB/s in 0.3s
2024-02-14 05:14:23 (151 KB/s) - ‘VF2_boot’ saved [52353/52353]
[riscv@fedora-riscv ~]$ ls
aarch64-boards
Fedora-RV64G-RPMS
Fedora-RV64G-RPMS.zip
rockchip-Add-initial-support-for-the-PinePhone-Pro.patch
rpi-Enable-using-the-DT-provided-by-the-Raspberry-Pi.patch
smbios-Simplify-reporting-of-unknown-values.patch
u-boot-2023.04.tar.bz2
uboot-tools-2023.04-1.fc38.src.rpm
uboot-tools.spec
uefi-distro-load-FDT-from-any-partition-on-boot-device.patch
VF2_boot
[riscv@fedora-riscv ~]$ rm VF2_boot
[riscv@fedora-riscv ~]$ ls
aarch64-boards
Fedora-RV64G-RPMS
Fedora-RV64G-RPMS.zip
rockchip-Add-initial-support-for-the-PinePhone-Pro.patch
rpi-Enable-using-the-DT-provided-by-the-Raspberry-Pi.patch
smbios-Simplify-reporting-of-unknown-values.patch
u-boot-2023.04.tar.bz2
uboot-tools-2023.04-1.fc38.src.rpm
uboot-tools.spec
uefi-distro-load-FDT-from-any-partition-on-boot-device.patch
[riscv@fedora-riscv ~]$ git clone https://github.com/riscv/opensbi.git
Cloning into 'opensbi'...
remote: Enumerating objects: 12163, done.
remote: Counting objects: 100% (12163/12163), done.
remote: Compressing objects: 100% (4289/4289), done.
remote: Total 12163 (delta 7665), reused 11183 (delta 7462), pack-reused 0
Receiving objects: 100% (12163/12163), 2.34 MiB | 1.30 MiB/s, done.
Resolving deltas: 100% (7665/7665), done.
[riscv@fedora-riscv ~]$

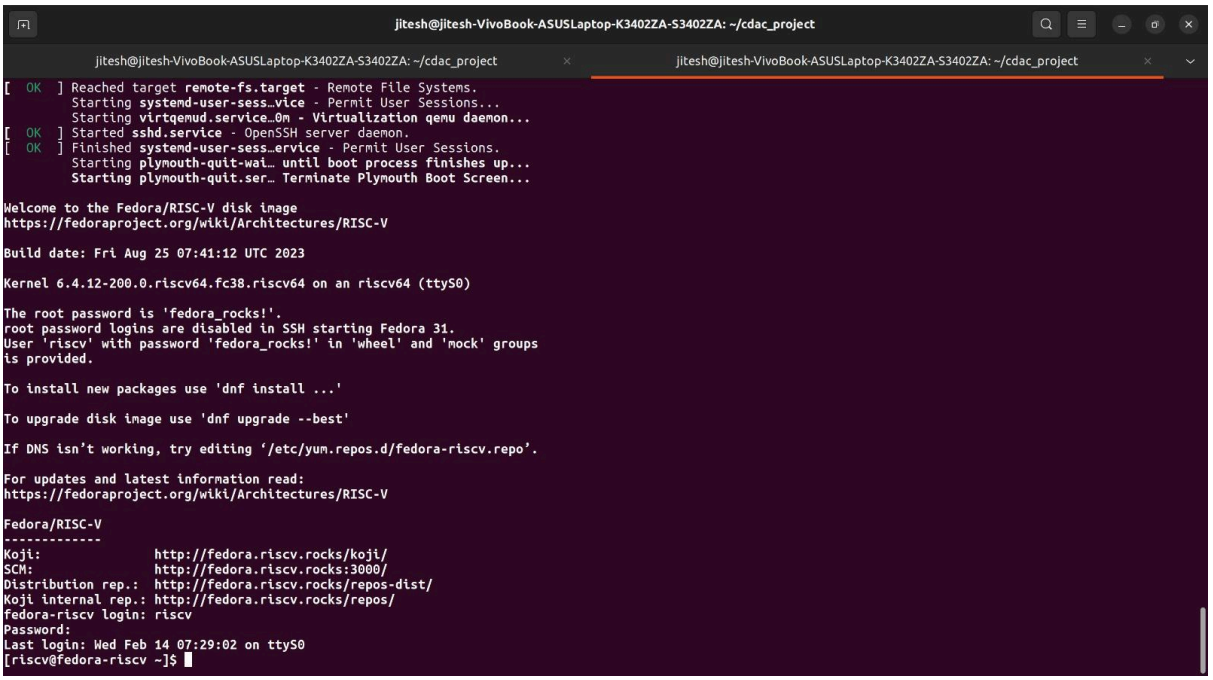

jitesh@jitesh-VivoBook-ASUSLaptop-K3402ZA-S3402ZA: ~/cdac_project
jitesh@jitesh-VivoBook-ASUSLaptop-K3402ZA-S3402ZA: ~/cdac_project
jitesh@jitesh-VivoBook-ASUSLaptop-K3402ZA-S3402ZA: ~/cdac_project
make: *** No targets specified and no makefile found. Stop.
[riscv@fedora-riscv ~]$ cd opensbi/
[riscv@fedora-riscv opensbi]$ ls
CONTRIBUTORS.md docs   include lib   platform   scripts
COPYING.BSD   firmware Kconfig Makefile README.md ThirdPartyNotices.md
[riscv@fedora-riscv opensbi]$ make OPENSBI=/home/riscv/opensbi/build/platform/generic/firmware/fw_dynamic.bin -j6
CC-DEP lib/sbi/riscv_asm.dep
CC-DEP lib/sbi/riscv_atomic.dep
AS-DEP lib/sbi/riscv_hardfp.dep
CC-DEP lib/sbi/riscv_locks.dep
CC-DEP lib/sbi/sbi_ecall.dep
GEN-DEP lib/sbi/sbi_ecall_exts.dep
GEN-DEP lib/sbi/sbi_ecall_exts.dep
CC-DEP lib/sbi/sbi_ecall_base.dep
CC-DEP lib/sbi/sbi_bitmap.dep
CC-DEP lib/sbi/sbi_bitops.dep
CC-DEP lib/sbi/sbi_console.dep
CC-DEP lib/sbi/sbi_domain.dep
CC-DEP lib/sbi/sbi_emulate_csr.dep
CC-DEP lib/sbi/sbi_fifo.dep
CC-DEP lib/sbi/sbi_hart.dep
CC-DEP lib/sbi/sbi_heap.dep
CC-DEP lib/sbi/sbi_math.dep
AS-DEP lib/sbi/sbi_hfence.dep
CC-DEP lib/sbi/sbi_hsm.dep
CC-DEP lib/sbi/sbi_illegal_insn.dep
CC-DEP lib/sbi/sbi_init.dep
CC-DEP lib/sbi/sbi_ipi.dep
CC-DEP lib/sbi/sbi_irqchip.dep
CC-DEP lib/sbi/sbi_misaligned_ldst.dep
CC-DEP lib/sbi/sbi_platform.dep
CC-DEP lib/sbi/sbi_pmu.dep
CC-DEP lib/sbi/sbi_dbtr.dep
CC-DEP lib/sbi/sbi_scratch.dep
CC-DEP lib/sbi/sbi_string.dep
CC-DEP lib/sbi/sbi_system.dep
CC-DEP lib/sbi/sbi_timer.dep
CC-DEP lib/sbi/sbi_tlb.dep
CC-DEP lib/sbi/sbi_trap.dep
CC-DEP lib/sbi/sbi_unpriv.dep


jitesh@jitesh-VivoBook-ASUSLaptop-K3402ZA-S3402ZA: ~/cdac_project
jitesh@jitesh-VivoBook-ASUSLaptop-K3402ZA-S3402ZA: ~/cdac_project
jitesh@jitesh-VivoBook-ASUSLaptop-K3402ZA-S3402ZA: ~/cdac_project
Your configuration changes were NOT saved.
[riscv@fedora-riscv u-boot-2023.04]$ ls
api cmd disk env Kbuild MAINTAINERS README
arch common doc examples Kconfig Makefile scripts
board config.mk drivers fs lib net test
boot configs dts include Licenses post tools
[riscv@fedora-riscv u-boot-2023.04]$ make OPENSBI=/home/riscv/opensbi/build/platform/generic/firmware/fw_dynamic.bin -j16
scripts/kconfig/conf --syncconfig Kconfig
***
*** Configuration file ".config" not found!
***
*** Please run some configurator (e.g. "make oldconfig" or
*** "make menuconfig" or "make xconfig").
***
make[2]: *** [scripts/kconfig/Makefile:75: syncconfig] Error 1
make[1]: *** [Makefile:575: syncconfig] Error 2
make: *** No rule to make target 'include/config/auto.conf', needed by 'include/config/uboot.release'. Stop.
[riscv@fedora-riscv u-boot-2023.04]$ make sifive_unleashed_defconfig
#
configuration written to .config
#
[riscv@fedora-riscv u-boot-2023.04]$


jitesh@jitesh-VivoBook-ASUSLaptop-K3402ZA-S3402ZA: ~/cdac_project
jitesh@jitesh-VivoBook-ASUSLaptop-K3402ZA-S3402ZA: ~/cdac_project
jitesh@jitesh-VivoBook-ASUSLaptop-K3402ZA-S3402ZA: ~/cdac_project
SHIPPED scripts/dtc/pylibfdt/libfdt.i
HOSTCC scripts/dtc/livetree.o
PYMOD rebuild
HOSTCC scripts/dtc/treesource.o
HOSTCC scripts/dtc/srcpos.o
HOSTCC scripts/dtc/checks.o
error: command 'swig' failed: No such file or directory
make[2]: *** [scripts/dtc/pylibfdt/Makefile:33: rebuild] Error 1
make[1]: *** [scripts/Makefile.build:397: scripts/dtc/pylibfdt] Error 2
make[1]: *** Waiting for unfinished jobs....
CC lib/asm-offsets.s
CC arch/riscv/lib/asm-offsets.s
UPD include/generated/asm-offsets.h
UPD include/generated/generic-asm-offsets.h
LDS u-boot.lds
make: *** [Makefile:2003: scripts_dtc] Error 2
[riscv@fedora-riscv u-boot-2023.04]$ pip install wheel
Defaulting to user installation because normal site-packages is not writeable
Collecting wheel
Downloading wheel-0.42.0-py3-none-any.whl (65 kB)
65.4/65.4 kB 723.0 kB/s eta 0:00:00
Installing collected packages: wheel
Successfully installed wheel-0.42.0
[riscv@fedora-riscv u-boot-2023.04]$


jitesh@jitesh-VivoBook-ASUSLaptop-K3402ZA-S3402ZA: ~/cdac_project
jitesh@jitesh-VivoBook-ASUSLaptop-K3402ZA-S3402ZA: ~/cdac_project
jitesh@jitesh-VivoBook-ASUSLaptop-K3402ZA-S3402ZA: ~/cdac_project
[riscv@fedora-riscv u-boot-2023.04]$ sudo dnf install swig
[sudo] password for riscv:
Last metadata expiration check: 0:06:21 ago on Wed 14 Feb 2024 05:35:11 AM EST.
Dependencies resolved.
Package Architecture Version Repository Size
Installing:
swig riscv64 4.1.1-4.fc38 fedora-riscv 1.5 M
Transaction Summary
Install 1 Package
Total download size: 1.5 M
Installed size: 5.6 M
Is this ok [y/N]: y
Downloading Packages:
swig-4.1.1-4.fc38.riscv64.rpm 631 kB/s | 1.5 MB 00:02
Total 623 kB/s | 1.5 MB 00:02
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
Preparing : 1/1
Installing : swig-4.1.1-4.fc38.riscv64 1/1
Running scriptlet: swig-4.1.1-4.fc38.riscv64 1/1
Verifying : swig-4.1.1-4.fc38.riscv64 1/1
Installed:
swig-4.1.1-4.fc38.riscv64
Complete!
[riscv@fedora-riscv u-boot-2023.04]$


jitesh@jitesh-VivoBook-ASUSLaptop-K3402ZA-S3402ZA: ~/cdac_project
jitesh@jitesh-VivoBook-ASUSLaptop-K3402ZA-S3402ZA: ~/cdac_project
jitesh@jitesh-VivoBook-ASUSLaptop-K3402ZA-S3402ZA: ~/cdac_project
jitesh@jitesh-VivoBook-ASUSLaptop-K3402ZA:~/cdac_project$ qemu-system-riscv64 -bios u-boot-spl.bin -nographic -machine virt -smp 6 -m 6G -device loader,file=u-boot.itb,addr=0x80200000 -object rng-random,filename=/dev/urandom,id=rng0 -device virtio-rng-device,rng=rng0 -device virtio-blk-device,drive=hd0 -drive file=Fedora-Developer-38-20230825.n.0-sda.raw,format=raw,id=hd0 -device virtio-net-device,netdev=usernet -netdev user,id=usernet,hostfwd=tcp::10000-:22


jitesh@jitesh-VivoBook-ASUSLaptop-K3402ZA-S3402ZA: ~/cdac_project
jitesh@jitesh-VivoBook-ASUSLaptop-K3402ZA-S3402ZA: ~/cdac_project
jitesh@jitesh-VivoBook-ASUSLaptop-K3402ZA-S3402ZA: ~/cdac_project
[OK] Reached target remote-fs.target - Remote File Systems.
Starting systemd-user-sess…vice - Permit User Sessions...
Starting virtqemud.service…0m - Virtualization qemu daemon...
[OK] Started sshd.service - OpenSSH server daemon.
[OK] Finished systemd-user-sess…ervice - Permit User Sessions.
Starting plymouth-quit-wai… until boot process finishes up...
Starting plymouth-quit.ser… Terminate Plymouth Boot Screen...
Welcome to the Fedora/RISC-V disk image
https://fedoraproject.org/wiki/Architectures/RISC-V
Build date: Fri Aug 25 07:41:12 UTC 2023
Kernel 6.4.12-200.0.riscv64.fc38.riscv64 on an riscv64 (ttyS0)
The root password is 'fedora_rocks!'.
root password logins are disabled in SSH starting Fedora 31.
User 'riscv' with password 'fedora_rocks!' in 'wheel' and 'mock' groups
is provided.
To install new packages use 'dnf install ...'
To upgrade disk image use 'dnf upgrade --best'
If DNS isn't working, try editing '/etc/yum.repos.d/fedora-riscv.repo'.
For updates and latest information read:
https://fedoraproject.org/wiki/Architectures/RISC-V
Fedora/RISC-V
Koji: http://fedora.riscv.rocks/koji/
SCM: http://fedora.riscv.rocks:3000/
Distribution rep.: http://fedora.riscv.rocks/repos-dist/
Koji internal rep.: http://fedora.riscv.rocks/repos/
fedora-riscv login: riscv
Password:
Last login: Wed Feb 14 07:29:02 on ttyS0
[riscv@fedora-riscv ~]$

7.2 Sifive Based EDK2 Implementation using QEMU

Currently, the open-source implementation of UEFI, EDK2, does not yet fully support RISC-V systems. Only Sifive has done some adaptations on its Sifive U540 RISC-V platforms. In QEMU, the Sifive U540 processor platform is booted with UEFI MODE using the QEMU-system-riscv64 command. QEMU-system-riscv64 command specifies CPU, virtual machine platform, firmware image, number of processers and memory size.

In this case, the CPU is Sifive-U540; the virtual machine platform is sifive_u; the firmware image is U540.fd which is compiled and generated by EDK2; the memory size is 2048M; the number of CPU cores is 5. The command above is able to start Linux, however, there are still three defects:

First, the command only support sifive-U540 platform in QEMU, which only simulate SiFive Unleashed boards. The more widely used virtio platform, which is hardware independent is not supported yet.

Second, modifying the memory size and the number of CPU cores of the virtual RISC-V system using single QEMU command is not supported. To modify firmware, one need to use the dump command in QEMU to export a new device tree file with modified firmware, and then copy the file to EDK2's directory to recompile and generate a new UEFI firmware image, and finally using the new firmware image to start QEMU.

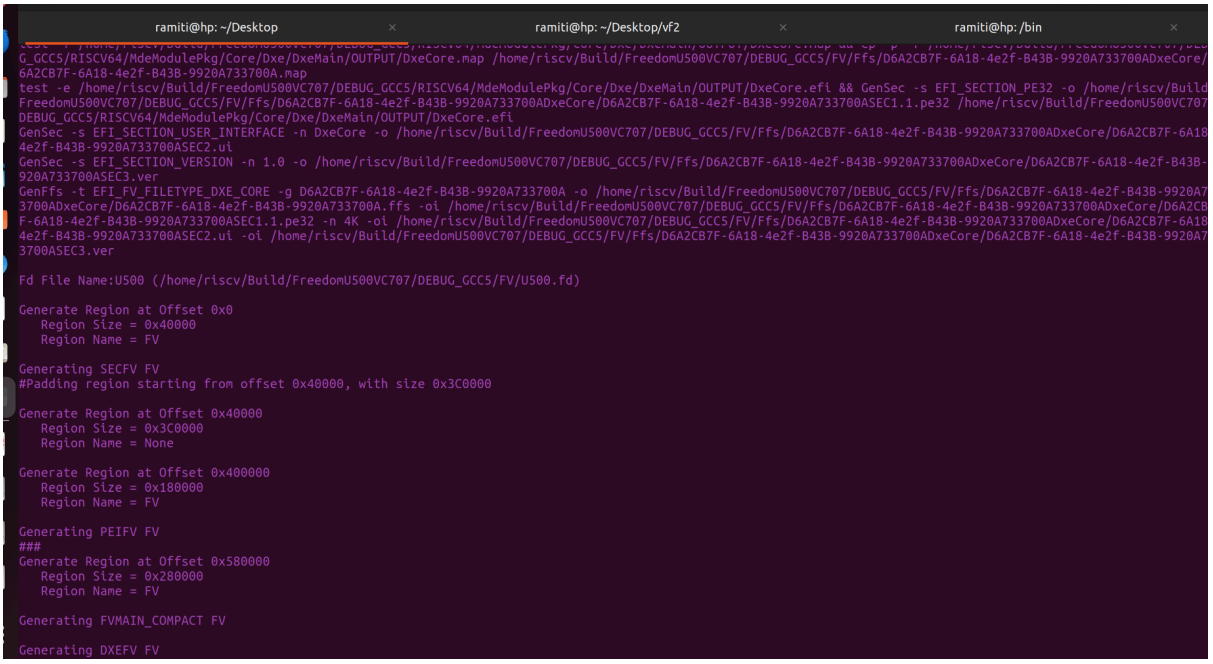
Third, booting to the operating system using one QEMU command is not yet possible. Only UEFI Shell could be started, and with the memory bias operations the system kernel and the file system pre-placed in the specified memory location are loaded, and finally the operating system can be started.

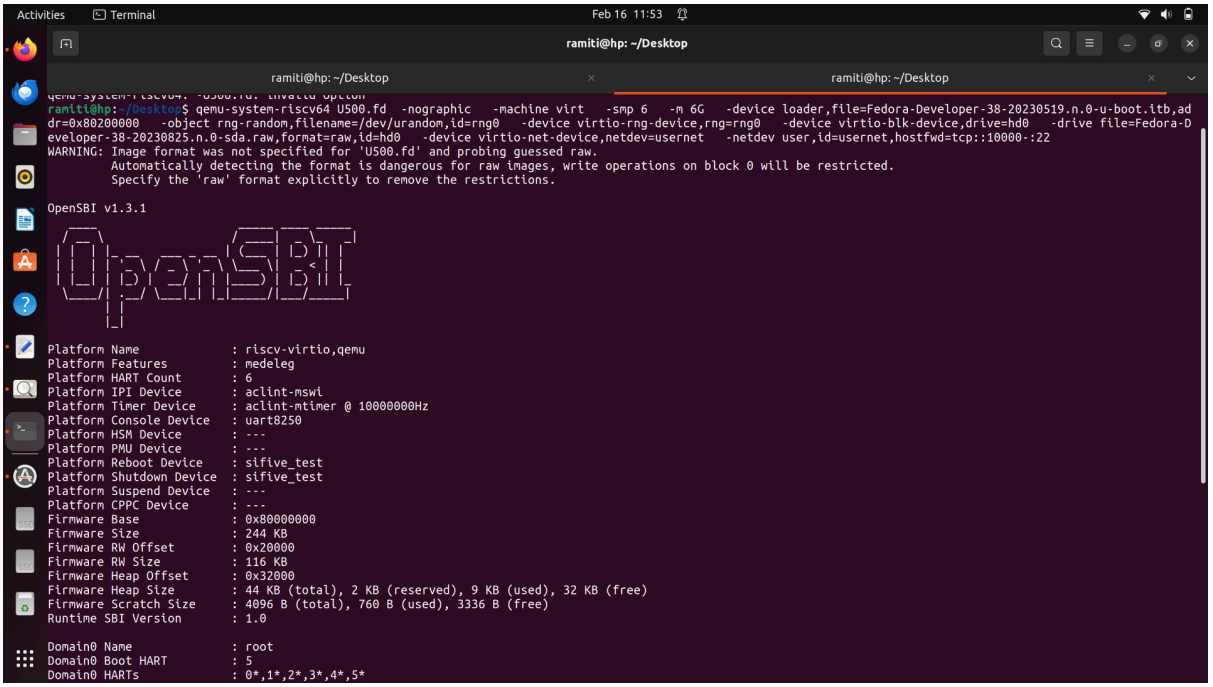
In light of the three defects mentioned above, this paper will expand the open source implementation of Sifive, so that it will support the general RISC-V virtual machine virtio under QEMU; support modifying the hardware of the simulation platform; support direct booting of the operating system.

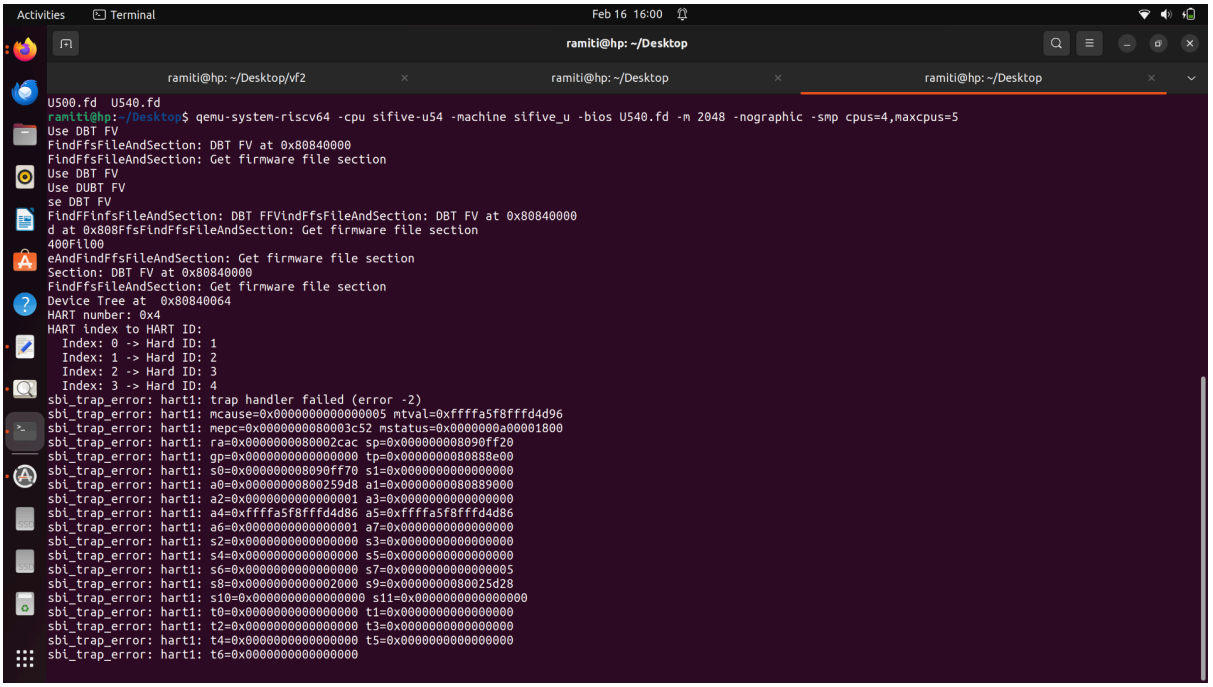
```
[riscv@fedora-riscv WORKSPACE]$ ls
[riscv@fedora-riscv WORKSPACE]$ git clone https://github.com/tianocore/edk2.git
Cloning into 'edk2'...
remote: Enumerating objects: 387655, done.
remote: Counting objects: 100% (285/285), done.
remote: Compressing objects: 100% (151/151), done.
remote: Total 387655 (delta 157), reused 215 (delta 129), pack-reused 387370
Receiving objects: 100% (387655/387655), 312.56 MiB | 2.18 MiB/s, done.
Resolving deltas: 100% (281507/281507), done.
Updating files: 100% (9129/9129), done.
[riscv@fedora-riscv WORKSPACE]$ git submodule update --init
fatal: not a git repository (or any of the parent directories): .git
[riscv@fedora-riscv WORKSPACE]$ git clone https://github.com/tianocore/edk2-platforms.git
Cloning into 'edk2-platforms'...
remote: Enumerating objects: 304607, done.
remote: Counting objects: 100% (5092/5092), done.
remote: Compressing objects: 100% (2163/2163), done.
Receiving objects:  17% (51784/304607), 33.22 MiB | 3.25 MiB/s
```

```
gcc  -c  -I .. -I ../Include/Common -I ../Include/ -I ../Include/IndustryStandard -I ../Common/ -I .. -I . -I ../../../../MdePkg/Include/RiscV64/ -I ../../../../
dePkg/Include -MD -fshort-wchar -fno-strict-aliasing -fwrapv -fno-delete-null-pointer-checks -Wall -Werror -Wno-deprecated-declarations -Wno-stringop-truncation
Wno-restrict -Wno-unused-result -nostdlib -g -O2  Decompress.c -o Decompress.o
gcc  -c  -I .. -I ../Include/Common -I ../Include/ -I ../Include/IndustryStandard -I ../Common/ -I .. -I . -I ../../../../MdePkg/Include/RiscV64/ -I ../../../../
dePkg/Include -MD -fshort-wchar -fno-strict-aliasing -fwrapv -fno-delete-null-pointer-checks -Wall -Werror -Wno-deprecated-declarations -Wno-stringop-truncation
Wno-restrict -Wno-unused-result -nostdlib -g -O2  EfiCompress.c -o EfiCompress.o
gcc  -c  -I .. -I ../Include/Common -I ../Include/ -I ../Include/IndustryStandard -I ../Common/ -I .. -I . -I ../../../../MdePkg/Include/RiscV64/ -I ../../../../
dePkg/Include -MD -fshort-wchar -fno-strict-aliasing -fwrapv -fno-delete-null-pointer-checks -Wall -Werror -Wno-deprecated-declarations -Wno-stringop-truncation
Wno-restrict -Wno-unused-result -nostdlib -g -O2  EfiUtilityMsgs.c -o EfiUtilityMsgs.o
gcc  -c  -I .. -I ../Include/Common -I ../Include/ -I ../Include/IndustryStandard -I ../Common/ -I .. -I . -I ../../../../MdePkg/Include/RiscV64/ -I ../../../../
dePkg/Include -MD -fshort-wchar -fno-strict-aliasing -fwrapv -fno-delete-null-pointer-checks -Wall -Werror -Wno-deprecated-declarations -Wno-stringop-truncation
Wno-restrict -Wno-unused-result -nostdlib -g -O2  FirmwareVolumeBuffer.c -o FirmwareVolumeBuffer.o
gcc  -c  -I .. -I ../Include/Common -I ../Include/ -I ../Include/IndustryStandard -I ../Common/ -I .. -I . -I ../../../../MdePkg/Include/RiscV64/ -I ../../../../
dePkg/Include -MD -fshort-wchar -fno-strict-aliasing -fwrapv -fno-delete-null-pointer-checks -Wall -Werror -Wno-deprecated-declarations -Wno-stringop-truncation
Wno-restrict -Wno-unused-result -nostdlib -g -O2  FvLib.c -o FvLib.o
gcc  -c  -I .. -I ../Include/Common -I ../Include/ -I ../Include/IndustryStandard -I ../Common/ -I .. -I . -I ../../../../MdePkg/Include/RiscV64/ -I ../../../../
dePkg/Include -MD -fshort-wchar -fno-strict-aliasing -fwrapv -fno-delete-null-pointer-checks -Wall -Werror -Wno-deprecated-declarations -Wno-stringop-truncation
Wno-restrict -Wno-unused-result -nostdlib -g -O2  MemoryFile.c -o MemoryFile.o
gcc  -c  -I .. -I ../Include/Common -I ../Include/ -I ../Include/IndustryStandard -I ../Common/ -I .. -I . -I ../../../../MdePkg/Include/RiscV64/ -I ../../../../
dePkg/Include -MD -fshort-wchar -fno-strict-aliasing -fwrapv -fno-delete-null-pointer-checks -Wall -Werror -Wno-deprecated-declarations -Wno-stringop-truncation
Wno-restrict -Wno-unused-result -nostdlib -g -O2  MyAlloc.c -o MyAlloc.o
gcc  -c  -I .. -I ../Include/Common -I ../Include/ -I ../Include/IndustryStandard -I ../Common/ -I .. -I . -I ../../../../MdePkg/Include/RiscV64/ -I ../../../../
dePkg/Include -MD -fshort-wchar -fno-strict-aliasing -fwrapv -fno-delete-null-pointer-checks -Wall -Werror -Wno-deprecated-declarations -Wno-stringop-truncation
Wno-restrict -Wno-unused-result -nostdlib -g -O2  OsPath.c -o OsPath.o
gcc  -c  -I .. -I ../Include/Common -I ../Include/ -I ../Include/IndustryStandard -I ../Common/ -I .. -I . -I ../../../../MdePkg/Include/RiscV64/ -I ../../../../
dePkg/Include -MD -fshort-wchar -fno-strict-aliasing -fwrapv -fno-delete-null-pointer-checks -Wall -Werror -Wno-deprecated-declarations -Wno-stringop-truncation
Wno-restrict -Wno-unused-result -nostdlib -g -O2  ParseGuidedSectionTools.c -o ParseGuidedSectionTools.o
gcc  -c  -I .. -I ../Include/Common -I ../Include/ -I ../Include/IndustryStandard -I ../Common/ -I .. -I . -I ../../../../MdePkg/Include/RiscV64/ -I ../../../../
dePkg/Include -MD -fshort-wchar -fno-strict-aliasing -fwrapv -fno-delete-null-pointer-checks -Wall -Werror -Wno-deprecated-declarations -Wno-stringop-truncation
Wno-restrict -Wno-unused-result -nostdlib -g -O2  ParseInf.c -o ParseInf.o
gcc  -c  -I .. -I ../Include/Common -I ../Include/ -I ../Include/IndustryStandard -I ../Common/ -I .. -I . -I ../../../../MdePkg/Include/RiscV64/ -I ../../../../
dePkg/Include -MD -fshort-wchar -fno-strict-aliasing -fwrapv -fno-delete-null-pointer-checks -Wall -Werror -Wno-deprecated-declarations -Wno-stringop-truncation
Wno-restrict -Wno-unused-result -nostdlib -g -O2  PeCoffLoaderEx.c -o PeCoffLoaderEx.o
gcc  -c  -I .. -I ../Include/Common -I ../Include/ -I ../Include/IndustryStandard -I ../Common/ -I .. -I . -I ../../../../MdePkg/Include/RiscV64/ -I ../../../../
dePkg/Include -MD -fshort-wchar -fno-strict-aliasing -fwrapv -fno-delete-null-pointer-checks -Wall -Werror -Wno-deprecated-declarations -Wno-stringop-truncation
Wno-restrict -Wno-unused-result -nostdlib -g -O2  SimpleFileParsing.c -o SimpleFileParsing.o
gcc  -c  -I .. -I ../Include/Common -I ../Include/ -I ../Include/IndustryStandard -I ../Common/ -I .. -I . -I ../../../../MdePkg/Include/RiscV64/ -I ../../../../
dePkg/Include -MD -fshort-wchar -fno-strict-aliasing -fwrapv -fno-delete-null-pointer-checks -Wall -Werror -Wno-deprecated-declarations -Wno-stringop-truncation
Wno-restrict -Wno-unused-result -nostdlib -g -O2  StringFuncs.c -o StringFuncs.o
gcc  -c  -I .. -I ../Include/Common -I ../Include/ -I ../Include/IndustryStandard -I ../Common/ -I .. -I . -I ../../../../MdePkg/Include/RiscV64/ -I ../../../../
dePkg/Include -MD -fshort-wchar -fno-strict-aliasing -fwrapv -fno-delete-null-pointer-checks -Wall -Werror -Wno-deprecated-declarations -Wno-stringop-truncation
Wno-restrict -Wno-unused-result -nostdlib -g -O2  TianoCompress.c -o TianoCompress.o
```

```
test_UPT_Xml_CommonXml (CheckPythonSyntax.Tests.test_UPT_Xml_CommonXml) ... ok
test_UPT_Xml_GuidProtocolPpiXml (CheckPythonSyntax.Tests.test_UPT_Xml_GuidProtocolPpiXml) ... ok
test_UPT_Xml_IniToXml (CheckPythonSyntax.Tests.test_UPT_Xml_IniToXml) ... ok
test_UPT_Xml_ModuleSurfaceAreaXml (CheckPythonSyntax.Tests.test_UPT_Xml_ModuleSurfaceAreaXml) ... ok
test_UPT_Xml_PackageSurfaceAreaXml (CheckPythonSyntax.Tests.test_UPT_Xml_PackageSurfaceAreaXml) ... ok
test_UPT_Xml_PcdXml (CheckPythonSyntax.Tests.test_UPT_Xml_PcdXml) ... ok
test_UPT_Xml_XmlParser (CheckPythonSyntax.Tests.test_UPT_Xml_XmlParser) ... ok
test_UPT_Xml_XmlParserMisc (CheckPythonSyntax.Tests.test_UPT_Xml_XmlParserMisc) ... ok
test_UPT_Xml___init__ (CheckPythonSyntax.Tests.test_UPT_Xml___init__) ... ok
test_Workspace_BuildClassObject (CheckPythonSyntax.Tests.test_Workspace_BuildClassObject) ... ok
test_Workspace_DecBuildData (CheckPythonSyntax.Tests.test_Workspace_DecBuildData) ... ok
test_Workspace_DscBuildData (CheckPythonSyntax.Tests.test_Workspace_DscBuildData) ... ok
test_Workspace_InfBuildData (CheckPythonSyntax.Tests.test_Workspace_InfBuildData) ... ok
test_Workspace_MetaDataTable (CheckPythonSyntax.Tests.test_Workspace_MetaDataTable) ... ok
test_Workspace_MetaFileCommentParser (CheckPythonSyntax.Tests.test_Workspace_MetaFileCommentParser) ... ok
test_Workspace_MetaFileParser (CheckPythonSyntax.Tests.test_Workspace_MetaFileParser) ... ok
test_Workspace_MetaFileTable (CheckPythonSyntax.Tests.test_Workspace_MetaFileTable) ... ok
test_Workspace_WorkspaceCommon (CheckPythonSyntax.Tests.test_Workspace_WorkspaceCommon) ... ok
test_Workspace_WorkspaceDatabase (CheckPythonSyntax.Tests.test_Workspace_WorkspaceDatabase) ... ok
test_Workspace___init__ (CheckPythonSyntax.Tests.test_Workspace___init__) ... ok
test_build_BuildReport (CheckPythonSyntax.Tests.test_build_BuildReport) ... ok
test_build___init__ (CheckPythonSyntax.Tests.test_build___init__) ... ok
test_build_build (CheckPythonSyntax.Tests.test_build_build) ... ok
test_build_buildoptions (CheckPythonSyntax.Tests.test_build_buildoptions) ... ok
test_sitecustomize (CheckPythonSyntax.Tests.test_sitecustomize) ... ok
test_tests_Split_test_split (CheckPythonSyntax.Tests.test_tests_Split_test_split) ... ok
test32bitUnicodeCharInUtf8Comment (CheckUnicodeSourceFiles.Tests.test32bitUnicodeCharInUtf8Comment) ... ok
test32bitUnicodeCharInUtf8File (CheckUnicodeSourceFiles.Tests.test32bitUnicodeCharInUtf8File) ... ok
testSupplementaryPlaneUnicodeCharInUtf16File (CheckUnicodeSourceFiles.Tests.testSupplementaryPlaneUnicodeCharInUtf16File) ... ok
testSurrogatePairUnicodeCharInUtf16File (CheckUnicodeSourceFiles.Tests.testSurrogatePairUnicodeCharInUtf16File) ... ok
testSurrogatePairUnicodeCharInUtf8File (CheckUnicodeSourceFiles.Tests.testSurrogatePairUnicodeCharInUtf8File) ... ok
testSurrogatePairUnicodeCharInUtf8FileWithBom (CheckUnicodeSourceFiles.Tests.testSurrogatePairUnicodeCharInUtf8FileWithBom) ... ok
testUtf16InUniFile (CheckUnicodeSourceFiles.Tests.testUtf16InUniFile) ... ok
testValidUtf8File (CheckUnicodeSourceFiles.Tests.testValidUtf8File) ... ok
testValidUtf8FileWithBom (CheckUnicodeSourceFiles.Tests.testValidUtf8FileWithBom) ... ok

----------------------------------------------------------------------
Ran 303 tests in 15.989s

OK
make[1]: Leaving directory '/home/riscv/WORKSPACE/edk2/BaseTools/Tests'
make: Leaving directory '/home/riscv/WORKSPACE/edk2/BaseTools'
[riscv@fedora-riscv WORKSPACE]$
```


ramiti@hp: ~/Desktop
ramiti@hp: ~/Desktop/vf2
ramiti@hp: /bin
G_GCC5/RISCV64/MdeModulePkg/Core/Dxe/DxeMain/OUTPUT/DxeCore.map /home/riscv/Build/FreedomU500VC707/DEBUG_GCC5/FV/Ffs/D6A2CB7F-6A18-4e2f-B43B-9920A733700ADxeCore/
6A2CB7F-6A18-4e2f-B43B-9920A733700A.map
test -e /home/riscv/Build/FreedomU500VC707/DEBUG_GCC5/RISCV64/MdeModulePkg/Core/Dxe/DxeMain/OUTPUT/DxeCore.efi && GenSec -s EFI_SECTION_PE32 -o /home/riscv/Build
FreedomU500VC707/DEBUG_GCC5/FV/Ffs/D6A2CB7F-6A18-4e2f-B43B-9920A733700ADxeCore/D6A2CB7F-6A18-4e2f-B43B-9920A733700ASEC1.1.pe32 /home/riscv/Build/FreedomU500VC707
DEBUG_GCC5/RISCV64/MdeModulePkg/Core/Dxe/DxeMain/OUTPUT/DxeCore.efi
GenSec -s EFI_SECTION_USER_INTERFACE -n DxeCore -o /home/riscv/Build/FreedomU500VC707/DEBUG_GCC5/FV/Ffs/D6A2CB7F-6A18-4e2f-B43B-9920A733700ADxeCore/D6A2CB7F-6A18
4e2f-B43B-9920A733700ASEC2.ui
GenSec -s EFI_SECTION_VERSION -n 1.0 -o /home/riscv/Build/FreedomU500VC707/DEBUG_GCC5/FV/Ffs/D6A2CB7F-6A18-4e2f-B43B-9920A733700ADxeCore/D6A2CB7F-6A18-4e2f-B43B-
920A733700ASEC3.ver
GenFfs -t EFI_FV_FILETYPE_DXE_CORE -g D6A2CB7F-6A18-4e2f-B43B-9920A733700A -o /home/riscv/Build/FreedomU500VC707/DEBUG_GCC5/FV/Ffs/D6A2CB7F-6A18-4e2f-B43B-9920A7
3700ADxeCore/D6A2CB7F-6A18-4e2f-B43B-9920A733700A.ffs -oi /home/riscv/Build/FreedomU500VC707/DEBUG_GCC5/FV/Ffs/D6A2CB7F-6A18-4e2f-B43B-9920A733700ADxeCore/D6A2CB
F-6A18-4e2f-B43B-9920A733700ASEC1.1.pe32 -n 4K -oi /home/riscv/Build/FreedomU500VC707/DEBUG_GCC5/FV/Ffs/D6A2CB7F-6A18-4e2f-B43B-9920A733700ADxeCore/D6A2CB7F-6A18
4e2f-B43B-9920A733700ASEC2.ui -oi /home/riscv/Build/FreedomU500VC707/DEBUG_GCC5/FV/Ffs/D6A2CB7F-6A18-4e2f-B43B-9920A733700ADxeCore/D6A2CB7F-6A18-4e2f-B43B-9920A7
3700ASEC3.ver
Fd File Name:U500 (/home/riscv/Build/FreedomU500VC707/DEBUG_GCC5/FV/U500.fd)
Generate Region at Offset 0x0
Region Size = 0x40000
Region Name = FV
Generating SECFV FV
#Padding region starting from offset 0x40000, with size 0x3C0000
Generate Region at Offset 0x40000
Region Size = 0x3C0000
Region Name = None
Generate Region at Offset 0x400000
Region Size = 0x180000
Region Name = FV
Generating PEIFV FV
###
Generate Region at Offset 0x580000
Region Size = 0x280000
Region Name = FV
Generating FVMAIN_COMPACT FV
Generating DXEFV FV


Activities
Terminal
Feb 16 11:53
ramiti@hp: ~/Desktop
ramiti@hp: ~/Desktop
ramiti@hp: ~/Desktop
ramiti@hp:~/Desktop$ qemu-system-riscv64 U500.fd -nographic -machine virt -smp 6 -m 6G -device loader,file=Fedora-Developer-38-20230519.n.0-u-boot.itb,ad
dr=0x80200000 -object rng-random,filename=/dev/urandom,id=rng0 -device virtio-rng-device,rng=rng0 -device virtio-blk-device,drive=hd0 -drive file=Fedora-D
eveloper-38-20230825.n.0-sda.raw,format=raw,id=hd0 -device virtio-net-device,netdev=usernet -netdev user,id=usernet,hostfwd=tcp::10000-:22
WARNING: Image format was not specified for 'U500.fd' 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.
OpenSBI v1.3.1
Platform Name : riscv-virtio,qemu
Platform Features : medeleg
Platform HART Count : 6
Platform IPI Device : aclint-mswi
Platform Timer Device : aclint-mtimer @ 10000000Hz
Platform Console Device : uart8250
Platform HSM Device : ---
Platform PMU Device : ---
Platform Reboot Device : sifive_test
Platform Shutdown Device : sifive_test
Platform Suspend Device : ---
Platform CPPC Device : ---
Firmware Base : 0x80000000
Firmware Size : 244 KB
Firmware RW Offset : 0x20000
Firmware RW Size : 116 KB
Firmware Heap Offset : 0x32000
Firmware Heap Size : 44 KB (total), 2 KB (reserved), 9 KB (used), 32 KB (free)
Firmware Scratch Size : 4096 B (total), 760 B (used), 3336 B (free)
Runtime SBI Version : 1.0
Domain0 Name : root
Domain0 Boot HART : 5
Domain0 HARTs : 0*,1*,2*,3*,4*,5*



7.3 Coreboot Implementation on QEMU

Coreboot Directory:

$ git clone https://review.coreboot.org/coreboot

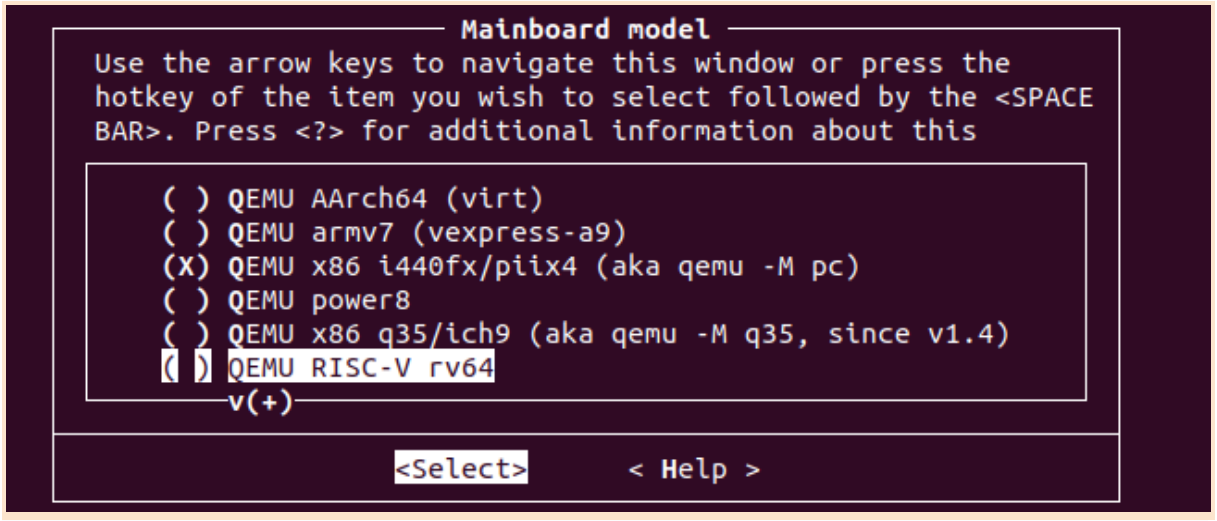
$ cd coreboot

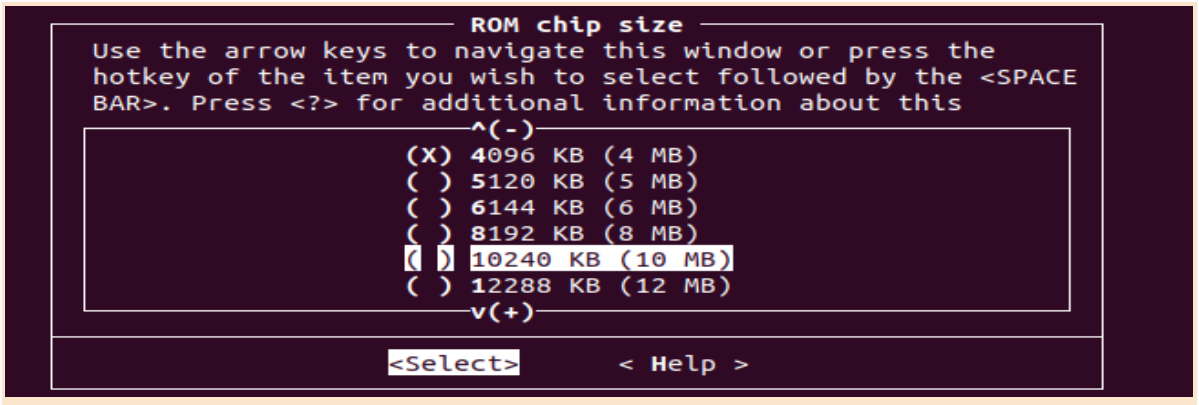
```
configuration written to /home/riscv/coreboot/.config

*** End of the configuration.
*** Execute 'make' to start the build or try 'make help'.

[riscv@fedora-riscv coreboot]$
[riscv@fedora-riscv coreboot]$ ls
3rdparty   configs         gnat.adc     Makefile       README.md   toolchain.inc
AUTHORS    COPYING         LICENSES     Makefile.inc   src         util
build      Documentation   MAINTAINERS  payloads       tests
[riscv@fedora-riscv coreboot]$ make savedefconfig
[riscv@fedora-riscv coreboot]$ ls
3rdparty   configs     Documentation   MAINTAINERS   payloads   tests
AUTHORS    COPYING     gnat.adc        Makefile      README.md  toolchain.inc
build      defconfig   LICENSES        Makefile.inc  src        util
[riscv@fedora-riscv coreboot]$ cat defconfig
CONFIG_BOARD_EMULATION_QEMU_RISCV_RV64=y
CONFIG_COREBOOT_ROMSIZE_KB_10240=y
[riscv@fedora-riscv coreboot]$
```

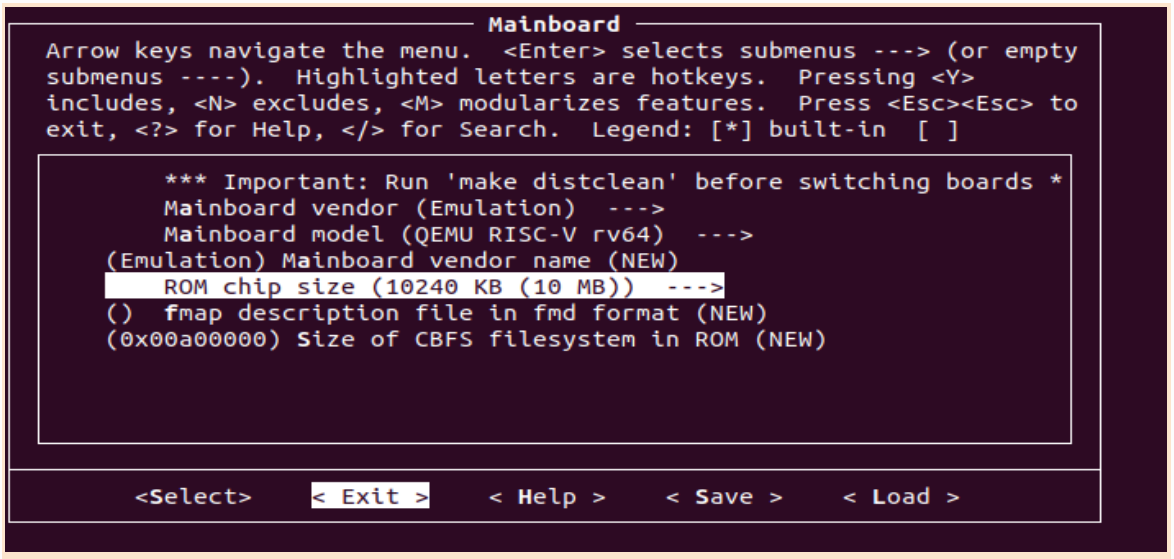
It is necessary to have configuration for RISC-V RV64.

Set Specifications For Coreboot:

$ make menuconfig





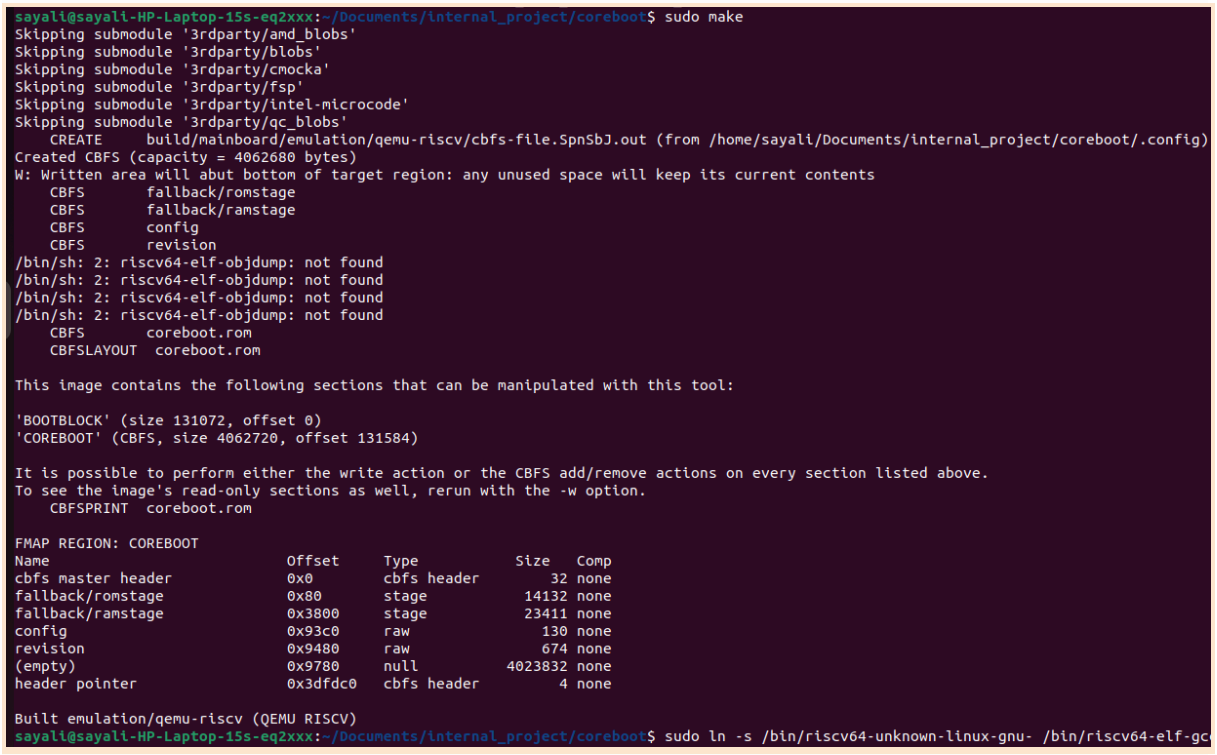


Emulation build using cross-toolchain:

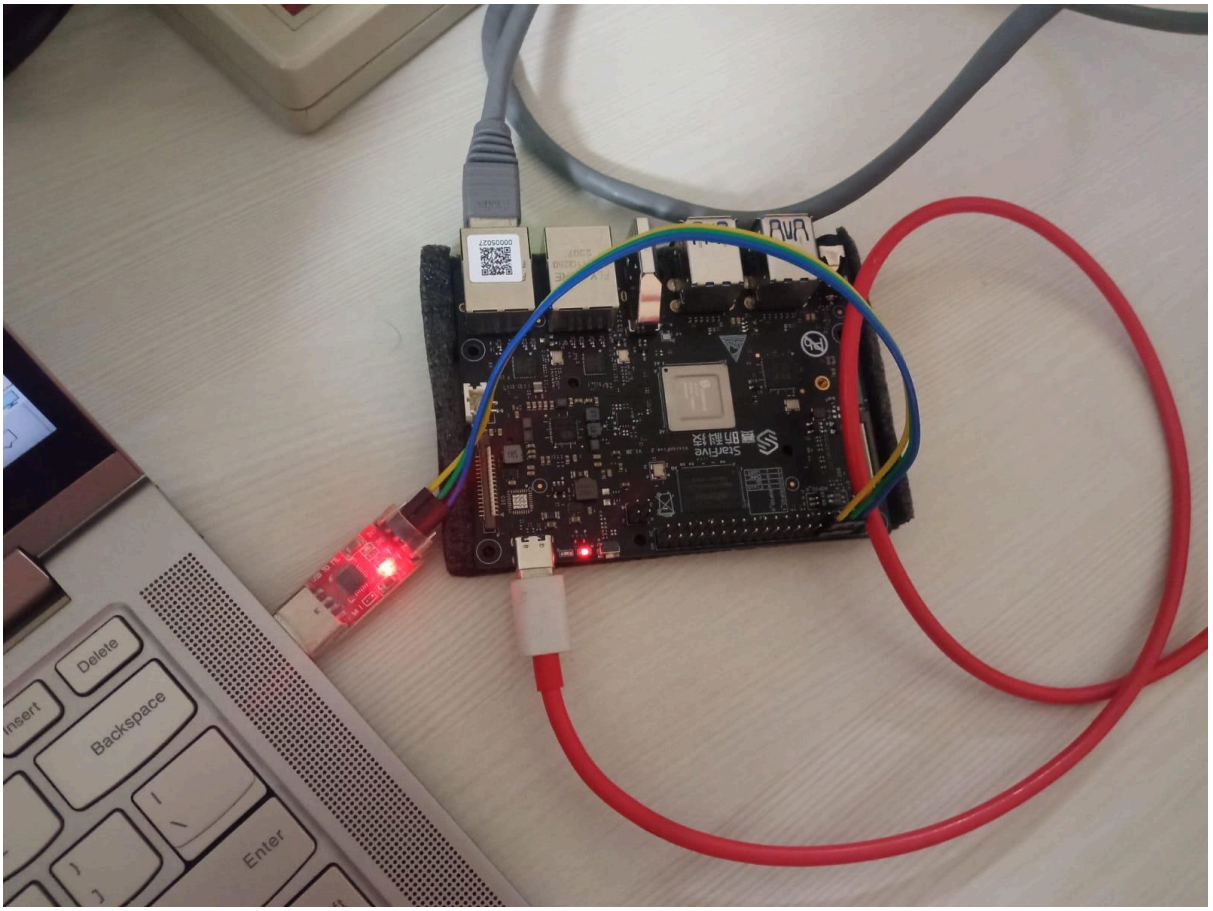
It will build an emulation for RISC-V Architecture.

$ sudo ln -s /bin/riscv64-unknown-linux-gnu-objdump /bin/riscv64-elf-objdump

$ sudo make



Chapter 8
Results




```
OpenSBI v0.9
   ____                    _____ ____ _____
  / __ \                  / ____|  _ \_   _|
 | |  | |_ __   ___ _ __ | (___ | |_) || |
 | |  | | '_ \ / _ \ '_ \ \___ \|  _ < | |
 | |__| | |_) |  __/ | | |____) | |_) || |_
  \____/| .__/ \___|_| |_|_____/|____/_____|
        | |
        |_|

Edk2OpensbiPlatformIrqchipInit: Entry
Edk2OpensbiPlatformIpiInit: Entry
Edk2OpensbiPlatformTlbrFlushLimit: Entry
Edk2OpensbiPlatformTimerInit: Entry
Platform Name             : StarFive VisionFive V2
Platform Features         : mfdeleg
Platform HART Count       : 4
Platform IPI Device       : aclint-mswi
Platform Timer Device     : aclint-mtimer
Platform Console Device   : uart8250
Platform HSM Device       : ---
Platform SysReset Device  : ---
Firmware Base             : 0x40000000
Firmware Size             : 256 KB
```

```
EDK IIlProtocolInterface: 752F3136-4E16-4FDC-A22A-E5F46812F4CA 7E9D5F98
UEFI v2.70 (EDK II, 0x00010000)008-7F9B-4F30-87AC-60C9FEF5DA4E 7E537350
Mapping table
      FS0: Alias(s):HD1d:;BLK4:
           VenHw(B549F005-4BD4-4020-A0CB-06F5478A68C3)/HD(3,GPT,022D130F-86D5-8742-9D64-FD20E727784
9,0xA000,0x32001)
     BLK0: Alias(s):
           VenHw(B549F005-4BD4-4020-A0CB-06F42BDA68C3)
     BLK1: Alias(s):
           VenHw(B549F005-4BD4-4020-A0CB-06F5478A68C3)
     BLK2: Alias(s):
           VenHw(B549F005-4BD4-4020-A0CB-06F5478A68C3)/HD(1,GPT,99EE9BC2-5F6B-AA4E-AD7D-38F26FC9A26
A,0x1000,0x1000)
     BLK3: Alias(s):
           VenHw(B549F005-4BD4-4020-A0CB-06F5478A68C3)/HD(2,GPT,C581D9A5-A9C3-F64F-94F2-D360CBA7B30
B,0x2000,0x8000)
     BLK5: Alias(s):
           VenHw(B549F005-4BD4-4020-A0CB-06F5478A68C3)/HD(4,GPT,0070AB6F-225D-9645-AE09-DDA7A26CF88
9,0x3C800,0x3AA3FDF)
Press ESC in 1 seconds to skip startup.nsh or any other key to continue.
InvalidateDataCacheRange:RISC-V unsupported function.
WriteBackDataCacheRange:RISC-V unsupported function.
InvalidateDataCacheRange:RISC-V unsupported function.
WriteBackDataCacheRange:RISC-V unsupported function.
```

```
      FS0: Alias(s):HD1d:;BLK4:
           VenHw(B549F005-4BD4-4020-A0CB-06F5478A68C3)/HD(3,GPT,022D130F-86D5-8742-9D64-FD20E727784
9,0xA000,0x32001)
     BLK0: Alias(s):
           VenHw(B549F005-4BD4-4020-A0CB-06F42BDA68C3)
     BLK1: Alias(s):
           VenHw(B549F005-4BD4-4020-A0CB-06F5478A68C3)
     BLK2: Alias(s):
           VenHw(B549F005-4BD4-4020-A0CB-06F5478A68C3)/HD(1,GPT,99EE9BC2-5F6B-AA4E-AD7D-38F26FC9A26
A,0x1000,0x1000)
     BLK3: Alias(s):
           VenHw(B549F005-4BD4-4020-A0CB-06F5478A68C3)/HD(2,GPT,C581D9A5-A9C3-F64F-94F2-D360CBA7B30
B,0x2000,0x8000)
     BLK5: Alias(s):
           VenHw(B549F005-4BD4-4020-A0CB-06F5478A68C3)/HD(4,GPT,0070AB6F-225D-9645-AE09-DDA7A26CF88
9,0x3C800,0x3AA3FDF)
Press ESC in 1 seconds to skip startup.nsh or any other key to continue.
InvalidateDataCacheRange:RISC-V unsupported function.
WriteBackDataCacheRange:RISC-V unsupported function.
InvalidateDataCacheRange:RISC-V unsupported function.
WriteBackDataCacheRange:RISC-V unsupported function.
Shell> ateDataCacheRange:RISC-V unsupported function.
WriteBackDataCacheRange:RISC-V unsupported function.
```

References

[1] Asanović, Krste; Patterson, David A. (6 August 2014). Instruction Sets Should Be Free: The Case For RISC-V (PDF). EECS Department, University of California, Berkeley. UCB/EECS-2014-146.

[2] Urquhart, Roddy (29 March 2021). "What Does RISC-V Stand For? A brief history of the open ISA". Systems & Design: Opinion. Semiconductor Engineering.

[3] Waterman, Andrew; Asanović, Krste (7 May 2017). "The RISC-V Instruction Set Manual, Volume I: Base User-Level ISA version 2.2" (PDF). RISC-V International. Retrieved 5 November 2021.

[4] Newsome, Tim; Wachs, Megan (22 March 2019). "RISC-V External Debug Support Version 0.13.2 d5029366d59e8563c08b6b9435f82573b603e48e" (PDF). RISC-V International. Retrieved 7 November 2021.

[5] About RISC-V, RISC-V International is the global non-profit home of the open standard RISC-V Instruction Set Architecture (ISA). RISC-V International.

[6] "RISC-V To Move HQ to Switzerland Amid Trade War Concerns". EE Times Europe. 28 November 2019.

[7] Demerjian, Chuck (7 August 2013). "A long look at how ARM licences chips: Part 1". SemiAccurate.

[8] Demerjian, Chuck (8 August 2013). "How ARM licences its IP for production: Part 2". SemiAccurate.

[9] Asanović, Krste. "Instruction Sets Should be Free" (PDF). U.C. Berkeley Technical Reports. Regents of the University of California. Retrieved 15 November 2016.

[10] "Rocket Core Generator". RISC-V. Regents of the University of California. Archived from the original on 6 October 2014. Retrieved 1 October 2014.

[11] Celio, Christopher; Love, Eric. "riscv-sodor: educational microarchitectures for risc-v isa". GitHub. Regents of the University of California. Retrieved 25 October 2019.

[12] "SHAKTI Processor Program". Indian Institute of Technology Madras. Retrieved 3 September 2019.

[13]https://doc.coreboot.org/arch/riscv/index.html “Coreboot Documentation”

[14]https://github.com/3mdeb “Github link”

[15]https://cdrdv2-public.intel.com/778593/778593_Technical%20Paper_coreboot_Integration_on_EGS_Rev0_9.pdf”Paper presentation”