

Project Report
On
Exploring various open-source boot loader
projects based on UEFI implementations for
RISC-V architecture



Submitted
In partial fulfilment
For the award of the Degree of

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

Under the esteemed guidance of: -
Mr. Rushikesh Jadhav
HPC Tech, C-DAC, ACTS (Pune)

Submitted By

Ashwini K. Darokar	240340130012
Rohini Sagar Baviskar	240340130036
Yadnesh Surendra Kayande	240340130054
Aboli Kiran Ahirrao	240340130001

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

ABSTRACT

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 **Exploring various open source boot loader project based on UEFI implementation for RISC-V architecture**. We express our deep gratitude towards her for inspiration, personal involvement, constructive criticism that she provided us along with technical guidance during 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 **Mrs. Risha P R (Program Head)** and **Mrs. 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.

Ashwini K. Darokar	240340130012
Rohini Sagar Baviskar	240340130036
Yadnesh Surendra Kayande	240340130054
Aboli Kiran Ahirrao	240340130001

Acknowledgement

The primary objective of this project is to explore and evaluate open source boot loaders that implement UEFI (Unified Extensible Firmware Interface) for RISC-V architecture, with a focus on integrating and extending their functionalities using Coreboot. The project aims to advance understanding and contribute to the development of boot loader solutions that enhance system initialization, configuration, and performance on RISC-V-based platforms.

The RISC-V architecture is an open standard instruction set architecture (ISA) that has gained significant traction for its flexibility and extensibility. As the ecosystem for RISC-V continues to grow, there is a need for robust and efficient boot loaders that support UEFI, a modern firmware interface standard that provides a range of functionalities including system initialization, hardware abstraction, and boot management.

Coreboot is an open-source project that aims to replace proprietary BIOS (Basic Input/Output System) firmware with a lightweight and flexible alternative. It provides a foundation for initializing hardware and loading additional software such as UEFI-compatible boot loaders

In parallel, the project delves into the intricacies of the boot process on RISC-V, providing insights into the architecture's specific requirements and optimizations. Furthermore, the project explores methods for transferring container images from emulated environments to physical devices, enabling seamless deployment of applications onto RISC-V-based embedded systems.

Table of Contents

S. No	Title	Page No.
	Front Page	I
	Acknowledgement	II
	Abstract	III
	Table of Contents	IV
1	Introduction	1-3
1.1	Introduction	1
1.2	Scope of work	2
1.3	Objectives	3
2	Why to used the coreboot for riscv	06
3	Methodology/ Techniques	7-23
3.1	Overview	7
3.2	Software Tools, Packages and Utilities	7
3.2.1	RISC-V Toolchain	7-8
3.2.2	Grub2	9
3.2.3	Coreboot	10
3.2.4	Fedora	12
3.2.5	QEMU	14
3.2.6	Parted	16

3.3	Methodology	21-23
4	Implementation	23-31
4.1	Coreboot	24
4.1.1	Building RISC-v Toolchain	24
4.1.4	Compiling Coreboot with Grub2	28
5	Results	32-
5.1	Results	34
6	Conclusion	35-36
6.1	Conclusion	
7	References	37
7.1	References	

Chapter 1

Introduction

1.1 Introduction

The emergence of RISC-V as an open and standardized instruction set architecture (ISA) has sparked significant interest in its application across various computing domains, including embedded systems. As RISC-V gains traction in the embedded space, the need for efficient deployment solutions becomes increasingly important. Containerization has emerged as a compelling approach for deploying applications, offering lightweight and scalable environments that can run across different platforms with minimal performance overhead.

This project focuses on composing disk and container images tailored for Linux on RISC-V, with a specific emphasis on future-generation embedded systems. The goal is to provide a comprehensive deployment solution that optimizes performance and resource utilization while maintaining the flexibility and portability of containerized applications.

The project leverages U-Boot as the bootloader, known for its flexibility and robustness in booting various operating systems on embedded devices. OpenSBI is used as the runtime, ensuring proper initialization and management of software on RISC-V platforms. The kernel is customized to include support for containers and virtualization, enabling efficient deployment and management of containerized applications. Additionally, the project integrates a root file system managed by dnf, a package manager for installing and managing software packages. This root file system is designed to resolve dependencies efficiently, simplifying the deployment process.

In addition to disk image composition, the project explores the intricacies of the boot process on RISC-V, providing insights into the architecture's specific requirements and

optimizations. Understanding these aspects is crucial for developing efficient and reliable boot procedures for RISC-V-based embedded systems.

Furthermore, the project investigates methods for transferring container images from emulated environments to physical devices. This capability is essential for deploying containerized applications onto RISC-V-based embedded systems, ensuring a seamless transition from development to deployment.

GRUB2 (GRand Unified Bootloader 2) is a highly versatile and widely used bootloader that serves as the intermediary between a computer's firmware and its operating system. As the successor to the original GRUB bootloader, GRUB2 introduces a host of improvements and new features, making it a preferred choice for many Linux distributions and other operating systems.

This project provides a comprehensive exploration of open-source boot loaders with UEFI implementation for RISC-V architecture. By analyzing these projects, we aim to gain insights into their functionality, performance, and potential improvements, contributing to the advancement of RISC-V-based systems.

.

1.2 Scope of Work

The scope of this project encompasses the creation of a comprehensive deployment solution for Linux on RISC-V architecture, specifically targeting future-generation embedded systems. The project will focus on development of firmware for RISC-V, with the aim of providing efficient and scalable deployment solutions. Key aspects of the project is to make a coreboot compatible with Grub2 for i386 and RISC-V.

This configuration is crucial for enabling RISC-V devices to boot Linux and load the operating system. Additionally, the project will integrate Grub2 as the runtime environment, ensuring proper initialization and management of software on RISC-V platforms.

The Linux kernel will be customized to include support for containers and virtualization, enabling efficient deployment and management of containerized applications. The project will also implement the coreboot which handovers control to Grub2 as a payload then the various OS selection menu should be displayed and user can select OS according to their requirement.

Performance testing and compatibility checks will be conducted to ensure that meet the requirements of future-generation embedded systems running on RISC-V architecture. Finally, the project will document the process, findings, and optimizations made during the project to contribute knowledge and insights to the RISC-V and containerization communities.

1.3 Objective

- To investigate and analyze the integration of UEFI (Unified Extensible Firmware Interface) with Coreboot firmware specifically for RISC-V architecture. This project aims to assess how Coreboot can be utilized in conjunction with UEFI to enhance boot management on RISC-V platforms, evaluate existing implementations, and identify areas for improvement.
- To update the kernel you have to build and flash a new image into the ROM chip.
- To change the kernel cmdline you have to build and flash a new image into the ROM chip.
- To use AMD VGA devices built coreboot with VGA Options ROM support.
- Try to minimize the kernel size by removing all features that won't be required on your hardware
- Booting from encrypted LUKS partitions
- Verifying signatures of files (interesting for initramfs and kernels)
- Running other coreboot payloads from [CBFS](#) (both compressed and uncompressed)

- **Implement dnf-based Root File System:** Integrate a root file system managed by dnf, enabling efficient resolution of dependencies and simplifying the deployment process of software packages on RISC-V devices.
- **Explore Boot Process Optimization:** Investigate and optimize the boot process on RISC-V architecture, ensuring efficient and reliable boot procedures for RISC-V- based embedded systems.
- **Develop Image Transfer Mechanism:** Explore methods for transferring qcow2 images from emulated environments to physical RISC-V devices, facilitating seamless deployment of applications.
- **Validate Performance and Compatibility:** Conduct performance testing and compatibility checks to ensure that the coreboot and qcow2 images meet the requirements of future-generation embedded systems running on RISC-V architecture.
- **Document and Share Findings:** Document the process, findings, and optimizations made during the project to contribute knowledge and insights to the RISC-V and containerization communities.

Chapter 2

Why to use Coreboot for Linux on RISC-V?

One of the current limitations in deploying Linux on RISC-V architecture is the lack of optimized deployment solutions tailored for future-generation embedded systems. Existing solutions may not fully leverage the capabilities of RISC-V or may suffer from performance overhead and compatibility issues. To overcome this limitation, this project focuses on creating firmware which is compatible for RISC-V, ensuring efficient deployment and management of applications.

Another limitation is the complexity of the boot process on RISC-V, which can impact system stability and performance. By integrating Coreboot as the bootloader as the runtime environment, the project aims to simplify and optimize the boot process, ensuring proper initialization and management of software on RISC-V platforms. This approach helps overcome the limitation of complex boot procedures and enhances system stability and performance using Grub2 as payload.

Furthermore, current deployment solutions may lack robust support for Grub on RISC-V, limiting the ability to efficiently deploy and manage applications. To address this limitation, the project customizes the operating system to include support for firmware and virtualization, enabling efficient deployment and

management of containerized applications on RISC-V devices. This approach enhances compatibility and scalability, overcoming the limitation of limited support on RISC-V.

Additionally, the project aims to simplify the deployment process of software packages on RISC-V devices. This simplification is expected to reduce complexity and improve the efficiency of deployment workflows, making it easier for developers to deploy applications on RISC-V devices.

Coreboot's role in optimizing firmware configurations, improving initialization processes, and supporting modern technologies, aligning with the needs of advanced embedded systems.

When GRUB is built for coreboot it looks for its runtime configuration in the file `etc/grub.cfg` within the CBFS. If this is missing it will provide a limited console only.

The file is an ordinary GRUB configuration file as specified in its documentation. It specifies menu entries and allows for quite some scripting.

Furthermore, the project seeks to contribute to the RISC-V communities by documenting the process, findings, and optimizations made during the project. By sharing this knowledge and insights, the project aims to advance the state of the art in these domains and foster collaboration among developers and researchers. This contribution is expected to benefit the broader community working on RISC-V and coreboot technologies, ultimately leading to the development of more efficient and reliable embedded systems solutions

Chapter 3

Methodology and Techniques

3.1 Overview

The methodology for exploring open-source boot loaders with UEFI implementation for RISC-V architecture related to Coreboot firmware involves several key stages: research, configuration, testing, and documentation. Each stage is designed to systematically evaluate how Coreboot integrates with UEFI for RISC-V platforms, assess performance, and identify areas for improvement.

The system architecture was designed with a focus on integrating Grub2 as the bootloader. This design consideration was crucial for ensuring compatibility, scalability, and performance optimization.

3.1.1) RISC-V Toolchain

The toolchain for this project primarily consisted of the RISC-V GNU Compiler Toolchain, which includes the GNU Compiler Collection (GCC) with support for RISC-V architecture. This toolchain provided the necessary compilers and libraries required for compiling the Grub2 bootloader and other software components for RISC-V architecture.

The RISC-V GNU Compiler Toolchain includes the following components:

- **GCC (GNU Compiler Collection):** GCC provides a suite of compilers for various programming languages, including C, C++, and assembly. For this project, GCC was used to compile the source code for U-Boot, OpenSBI, the Linux kernel, and other software components to generate executable binaries for RISC-V architecture.
- **Binutils:** Binutils is a collection of binary utilities, including an assembler, linker, and other tools for manipulating binary files. Binutils was used in conjunction with GCC to assemble and link the compiled code into executable binaries for RISC-V architecture.
- **Libraries:** The toolchain includes standard C libraries (e.g., glibc) and other libraries required for compiling and linking software for RISC-V architecture. These libraries provide essential functions and utilities that are used by the compiled code.
- **Cross-Compiler:** The toolchain also includes a cross-compiler for RISC-V architecture, which allows compiling software on a host system (e.g., x86) for execution on a target RISC-V platform. This cross-compiler ensures that the compiled binaries are compatible with the RISC-V instruction set architecture (ISA).
- **Debugging and Profiling Tools:** GCC also includes debugging and profiling tools, such as GDB (GNU Debugger), which were used for debugging and profiling the compiled software components. These tools helped identify and fix issues in the code, ensuring the reliability and performance of the deployment solution.

Overall, the RISC-V GNU Compiler Toolchain provided a comprehensive set of compilers and libraries necessary for compiling and building the software components for Linux on RISC-V architecture. The toolchain ensured compatibility, efficiency, and

reliability in the development process, contributing to the successful completion of the project.

3.1.2) Grub2

GNU GRUB is a [Multiboot](#) boot loader. It was derived from GRUB, the **GRand Unified Bootloader**, which was originally designed and implemented by Erich Stefan Boleyn.

Briefly, a boot loader is the first software program that runs when a computer starts. It is responsible for loading and transferring control to the operating system kernel software (such as [the Hurd](#) or Linux). The kernel, in turn, initializes the rest of the operating system (e.g. GNU).

GRUB 2 has replaced what was formerly known as GRUB (i.e. version 0.9x), which has, in turn, become [GRUB Legacy](#). Enhancements to GRUB are still being made, but the current released versions are quite usable for normal operation.

GRUB2 is the latest version of **GNU GRUB**, the **GRand Unified Bootloader**. A boot loader is the first software program that runs when a computer starts. It is responsible for loading and transferring control to the operating system kernel. In Fedora, the kernel is Linux. The kernel then initializes the rest of the operating system.

GRUB2 is the follower of the previous version **GRUB** (version 0.9x). The original version is available under the name **GRUB Legacy**.

Since Fedora 16, **GRUB2** has been the default bootloader on x86 BIOS systems. For upgrades of BIOS systems, the default is also to install **GRUB2**, but you can opt to skip bootloader configuration entirely.

Under EFI, **GRUB2** looks for its configuration in `/boot/efi/EFI/fedora/grub.cfg`, however the postinstall script of grub2-common installs a small shim which chains to the standard configuration at `/boot/grub2/grub.cfg` which is generated above. To reset this shim to defaults, delete the existing `/boot/efi/EFI/fedora/grub.cfg` and then reinstall grub2-common.

3.1.3) Coreboot

coreboot, formerly known as **LinuxBIOS**,^[5] 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.

One of the coreboot variants is Libreboot, a software distribution partly free of proprietary blobs, aimed at end users.

CPU architectures supported by coreboot include IA-32, x86-64, ARM, ARM64, MIPS and RISC-V. Artec Group added Geode LX support for its ThinCan model DBE61; that code was adopted by AMD and further improved for the OLPC after it was upgraded to the Geode LX platform, and is further developed by the coreboot community to support other Geode variants. coreboot can be flashed onto a Geode platform using Flashrom.

3.1.4) Fedora

Fedora, also known as Fedora Linux, is a popular Open Source Linux-based operating system (OS). Designed as a secure, general-purpose OS, Fedora is developed on a six-month to eight-month release cycle under the Fedora Project. Both the OS and the Fedora Project are financially sponsored and supported by Red Hat.

Fedora is a free and open source OS platform for hardware, clouds and containers. Based on the Linux OS kernel architecture, Fedora Linux is a user-friendly OS that enables users to perform their tasks easily and efficiently with minimal friction. The name Fedora refers to the characteristic fedora hat in Red Hat's Shadowman logo.

3.1.5) QEMU

QEMU is a machine emulator that can run operating systems and programs for one machine on a different machine. However, it is more often used as a virtualiser in collaboration with KVM kernel components. In that case it uses the hardware virtualisation technology to virtualise guests.

- **QEMU Overview:** QEMU (Quick EMUlator) is an open-source emulator and virtualization tool that allows for the emulation of various hardware architectures. It provides a platform for running operating systems and software in a virtual environment, making it valuable for development, testing, and debugging.
- **Features:** QEMU supports a wide range of hardware architectures, including RISC-V. It offers features such as full system emulation, user-space emulation, and support for various peripherals and devices.
- **RISC-V Support:** QEMU includes support for RISC-V architecture, enabling the emulation of RISC-V processors and systems. This allows developers to test and develop software for RISC-V without needing physical hardware.
- **Use Cases:** QEMU is used for testing operating systems, boot loaders, and firmware in a controlled virtual environment, making it a useful tool for projects involving Coreboot and UEFI on RISC-V.

3.1.6) Parted

Parted: Parted is a command-line utility for creating, resizing, and managing disk partitions. It supports various disk partitioning schemes, including MBR (Master Boot Record) and GPT (GUID Partition Table), making it a versatile tool for disk management tasks on Linux systems.

Key Features of Parted:

- **Create Partitions:** Parted allows users to create new disk partitions, specifying

the partition type, size, and file system format.

- **Resize Partitions:** Parted can resize existing partitions, allowing users to increase or decrease the size of a partition without losing data.
- **Move and Copy Partitions:** Parted can move and copy partitions to different locations on the disk, adjusting the partition table accordingly.
- **Delete Partitions:** Parted can delete partitions, removing them from the partition table and freeing up the disk space.
- **Check and Repair Partitions:** Parted can check the integrity of partition tables and partitions, and repair any errors found.
- **Usage of Parted:** Parted is used from the command line, with options and arguments specified to perform specific partitioning tasks. For example, to create a new partition, the `mkpart` command is used with parameters specifying the start and end of the partition, the partition type, and the file system format. Parted also provides an interactive mode, where users can issue commands and see the effects on the disk partition table in real-time.
- **Caution with Parted:** While parted is a powerful tool for managing disk partitions, it is important to use it with caution, as incorrect use can result in data loss. It is recommended to back up important data before making any changes to disk partitions.

3.2 Methodology

The methodology for the project involving the compilation of Coreboot the kernel with firmware support, and the creation of a root file system RISC-V systems can be outlined as follows:

- **Setting Up the Build Environment:** Install necessary tools and dependencies for building Coreboot for i386 and RISC-V. This may include compilers, build systems, and libraries required for RISC-V development.
- **Compiling COREBOOT:** Obtain the COREBOOT source code from the repository or a specific
<https://review.coreboot.org/coreboot>
- . Configure Coreboot for the target RISC-V platform, enabling any required features and settings. Compile Coreboot using the appropriate cross- compilation toolchain for RISC-V.
- **Creating the GRUB File System (grub.cfg):** Choose a base distribution for the RootFS, such as Fedora, Debian, Customize the RootFS configuration to include only the necessary components and packages for your application. Build the RootFS using the chosen build system, ensuring that it includes the required libraries, utilities, and configurations for container support.

Use tools like qemu-img to create the , ensuring that it is properly formatted and contains the necessary components for booting and running the system.

- **Documentation and Reporting:** Document the entire build process, including any custom configurations or settings. Provide instructions for building and deploying the coreboot on other RISC-V systems. Prepare a detailed report summarizing the methodology, results, and any issues encountered during the project

Chapter 4

Implementation

4.1) Coreboot:

Prerequisites: Fedora Operating System , Fedora image in formate of qcow2.

4.1.1) Building Toolchain:

Step 1: `git clone https://review.coreboot.org/coreboot`

Step 2: `./bootstrap`

Step 3 : `./configure -platform=coreboot -prefix=/home/desd/grub_install`

Step 4: `make -j8`

Step 5: `sudo make install`

Step 6: `qemu-system-x86_64-drive file=Fedora-Server-KVM-Rawhide-20240730.n.0.x86_64.qcow2,format=qcow2 -m 8G -smp 7 -nographic`

Description:

- Step 1:** Clone the Coreboot Repository. Use the git clone command to clone the Toolchain repository from GitHub. This repository contains the source code for the Coreboot compiler and related tools.

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

- Step 2:** Install Required Dependencies. Use the package manager to install the required dependencies for building the toolchain. The dependencies include autoconf, automake, libmpc-devel, mpfr-devel, freetype-devel bison, flex, texinfo, patchutils, gcc, gcc-c++ and expat-devel.

Example:

```
qemu-system-x86_64-drive file=Fedora-Server-KVM-Rawhide-  
20240730.n.0.x86_64.qcow2,format=qcow2 -m 8G -smp 7 -nographic
```

Step 3: Configure the Toolchain. Run the configure script in the Coreboot Toolchain directory to configure the build settings. Specify the installation prefix using the --prefix option (e.g., /opt/coreboot) to specify the path where the toolchain will be installed. Use the --with-arch option to specify the Coreboot architecture and ABI (Application Binary Interface) to target.

Example:

```
./configure -platform=coreboot -prefix=/home/desd/grub_install
```

Step 4: Build the Toolchain . Use the make command to build the Coreboot Toolchain for Linux. The make command will compile and other tools necessary for building Coreboot binaries.

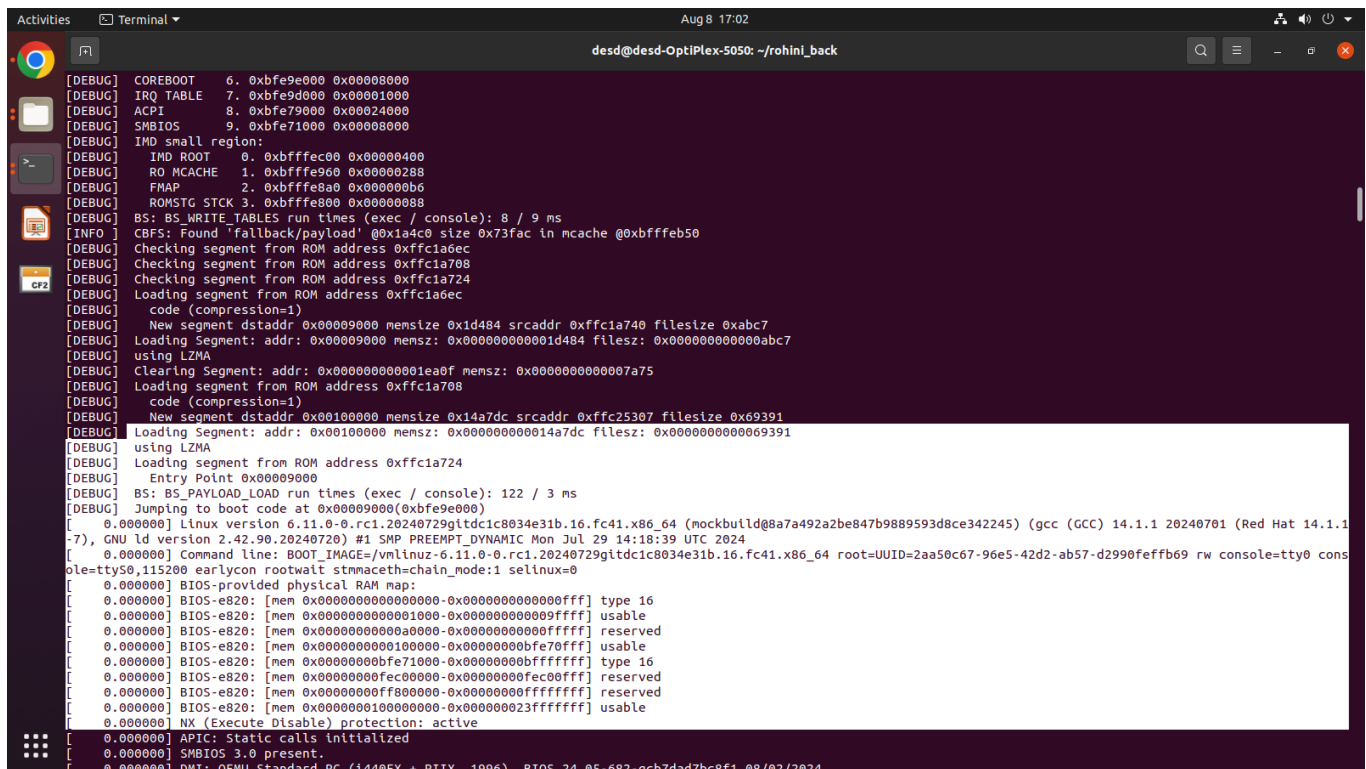
Example: make -j8

After completing these steps, you should have a compiled Coreboot installed in the specified prefix directory (grub_install in this case). This toolchain can be used to compile and build Coreboot binaries for the specified architecture i386 and ABI in grub_install/sbin/grub.cfg.

Chapter 5

Results

Coreboot compiled for i386

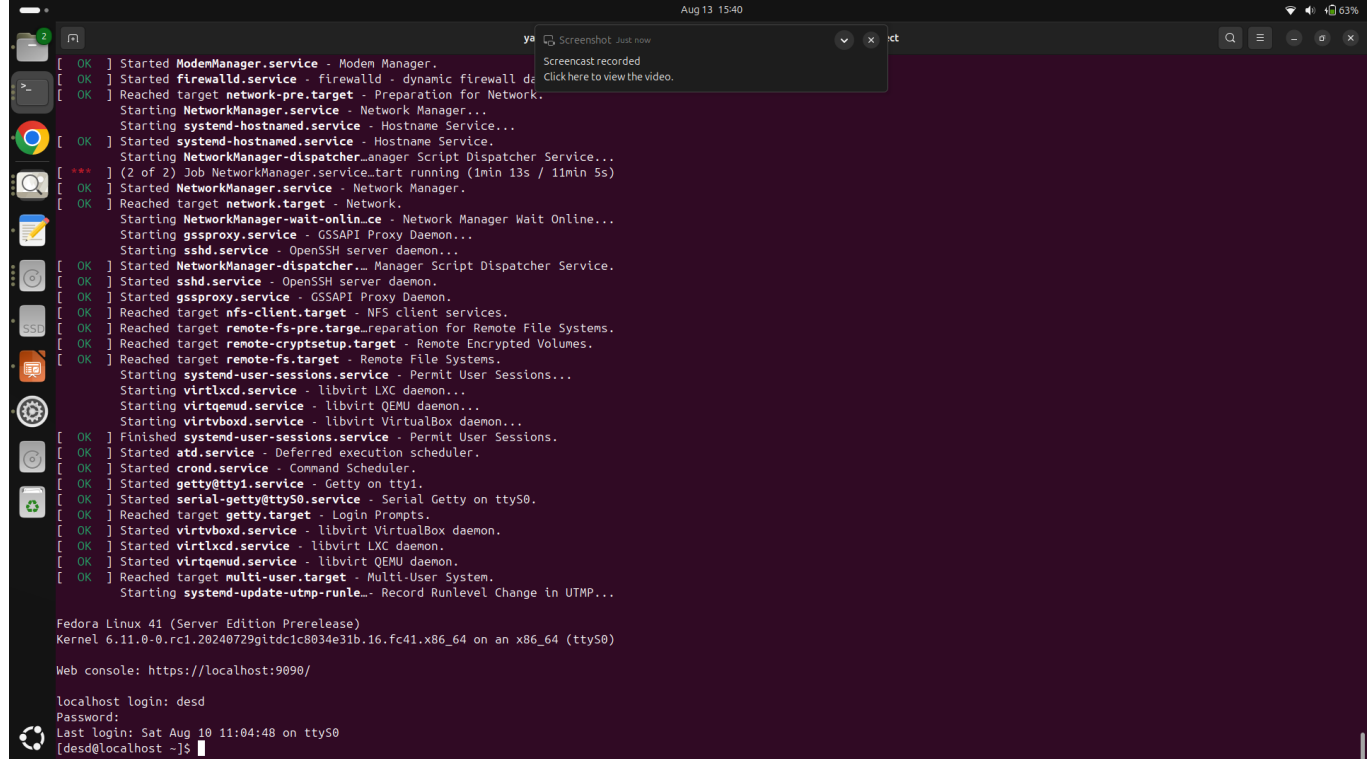


```
[DEBUG] COREBOOT 6. 0xbfe9e000 0x00000000
[DEBUG] IRQ TABLE 7. 0xbfe9d000 0x00001000
[DEBUG] ACPI 8. 0xbfe79000 0x00024000
[DEBUG] SMBIOS 9. 0xbfe71000 0x00000000
[DEBUG] IN0 small region:
[DEBUG] IN0 ROOT 0. 0xbfffec00 0x00000400
[DEBUG] RO MCACHE 1. 0xbfffe900 0x00000288
[DEBUG] FMAP 2. 0xbfffe8a0 0x000000b6
[DEBUG] ROMSTG STCK 3. 0xbfffe800 0x00000088
[DEBUG] BS: BS_WRITE_TABLES run times (exec / console): 8 / 9 ms
[INFO] CBFS: Found 'fallback/payload' @0x1a4c0 size 0x73fac in mcache @0xbfffeb50
[DEBUG] Checking segment from ROM address 0xffc1a6ec
[DEBUG] Checking segment from ROM address 0xffc1a708
[DEBUG] Checking segment from ROM address 0xffc1a724
[DEBUG] Loading segment from ROM address 0xffc1a6ec
[DEBUG] code (compression=1)
[DEBUG] New segment dstaddr 0x00009000 memsize 0x1d484 srcaddr 0xffc1a740 filesize 0xabc7
[DEBUG] Loading Segment: addr: 0x00009000 memsz: 0x000000000001d484 filesz: 0x000000000000abc7
[DEBUG] using LZMA
[DEBUG] Clearing Segment: addr: 0x000000000001ea0f memsz: 0x0000000000007a75
[DEBUG] Loading segment from ROM address 0xffc1a708
[DEBUG] code (compression=1)
[DEBUG] New segment dstaddr 0x00100000 memsize 0x14a7dc srcaddr 0xffc25307 filesize 0x69391
[DEBUG] Loading Segment: addr: 0x00100000 memsz: 0x0000000000014a7dc filesz: 0x00000000000069391
[DEBUG] using LZMA
[DEBUG] Loading segment from ROM address 0xffc1a724
[DEBUG] Entry Point 0x00009000
[DEBUG] BS: BS_PAYLOAD_LOAD run times (exec / console): 122 / 3 ms
[DEBUG] Jumping to boot code at 0x00009000(0xbfe9e000)
[ 0.000000] Linux version 6.11.0-0.rc1.20240729gitdclc8034e31b.16.fc41.x86_64 (mockbuild@8a7a492a2be847b9889593d8ce342245) (gcc (GCC) 14.1.1 20240701 (Red Hat 14.1.1
-7), GNU ld version 2.42.90.20240720) #1 SMP PREEMPT_DYNAMIC Mon Jul 29 14:18:39 UTC 2024
[ 0.000000] Command line: BOOT_IMAGE=/vmlinuz-6.11.0-0.rc1.20240729gitdclc8034e31b.16.fc41.x86_64 root=UUID=2aa50c67-96e5-42d2-ab57-d2990feffb69 rw console=tty0 cons
ole=ttyS0,115200 earlycon rootwait stmmaceth=chain_mode:1 selinux=0
[ 0.000000] BIOS-provided physical RAM map:
[ 0.000000] BIOS-e820: [mem 0x0000000000000000-0x0000000000000fff] type 16
[ 0.000000] BIOS-e820: [mem 0x0000000000001000-0x0000000000009fff] usable
[ 0.000000] BIOS-e820: [mem 0x000000000000a000-0x000000000000ffff] reserved
[ 0.000000] BIOS-e820: [mem 0x0000000000010000-0x000000000000bfe70fff] usable
[ 0.000000] BIOS-e820: [mem 0x00000000000bfe71000-0x00000000000bffffff] type 16
[ 0.000000] BIOS-e820: [mem 0x000000000fec00000-0x000000000fec00fff] reserved
[ 0.000000] BIOS-e820: [mem 0x00000000ff800000-0x00000000ffffff] reserved
[ 0.000000] BIOS-e820: [mem 0x0000000010000000-0x0000000023ffffff] usable
[ 0.000000] NX (Execute Disable) protection: active
[ 0.000000] APIC: Static calls initialized
[ 0.000000] SMBIOS 3.0 present.
[ 0.000000] DMI: OEMU Standard PC (i440FX + PITX - 1996) BIOS 24.05-682-qcb7dad7bcbf1 08/02/2024
```

Fig: Jumping of payload for i836

Exploring various open source boot loader project based on UEFI implementation for RISC-V architecture

S



```
[ OK ] Started ModemManager.service - Modem Manager.
[ OK ] Started firewalld.service - firewalld - dynamic firewall de
[ OK ] Reached target network-pre.target - Preparation for Network.
Starting NetworkManager.service - Network Manager...
Starting systemd-hostnamed.service - Hostname Service...
[ OK ] Started systemd-hostnamed.service - Hostname Service.
Starting NetworkManager-dispatcher.service - Network Manager...
[ *** ] (2 of 2) Job NetworkManager.service.start running (1min 13s / 11min 5s)
[ OK ] Started NetworkManager.service - Network Manager.
Reached target network.target - Network.
Starting NetworkManager-wait-online.service - Network Manager Wait Online...
Starting gssproxy.service - GSSAPI Proxy Daemon...
Starting sshd.service - OpenSSH server daemon...
[ OK ] Started NetworkManager-dispatcher.service - Network Manager Script Dispatcher Service.
[ OK ] Started sshd.service - OpenSSH server daemon.
[ OK ] Started gssproxy.service - GSSAPI Proxy Daemon.
Reached target nfs-client.target - NFS client services.
Reached target remote-fs-pre.target - Preparation for Remote File Systems.
Reached target remote-cryptsetup.target - Remote Encrypted Volumes.
Reached target remote-fs.target - Remote File Systems.
Starting systemd-user-sessions.service - Permit User Sessions...
Starting virtlxc.service - libvirt LXC daemon...
Starting virtqemu.service - libvirt QEMU daemon...
Starting virtvboxd.service - libvirt VirtualBox daemon...
[ OK ] Finished systemd-user-sessions.service - Permit User Sessions.
[ OK ] Started atd.service - Deferred execution scheduler.
[ OK ] Started crond.service - Command Scheduler.
[ OK ] Started getty@tty1.service - Getty on tty1.
[ OK ] Started serial-getty@ttyS0.service - Serial Getty on ttyS0.
[ OK ] Reached target getty.target - Login Prompts.
[ OK ] Started virtvboxd.service - libvirt VirtualBox daemon.
[ OK ] Started virtlxc.service - libvirt LXC daemon.
[ OK ] Started virtqemu.service - libvirt QEMU daemon.
Reached target multi-user.target - Multi-User System.
Starting systemd-update-utmp-runlevel.service - Record Runlevel Change in UTMP...

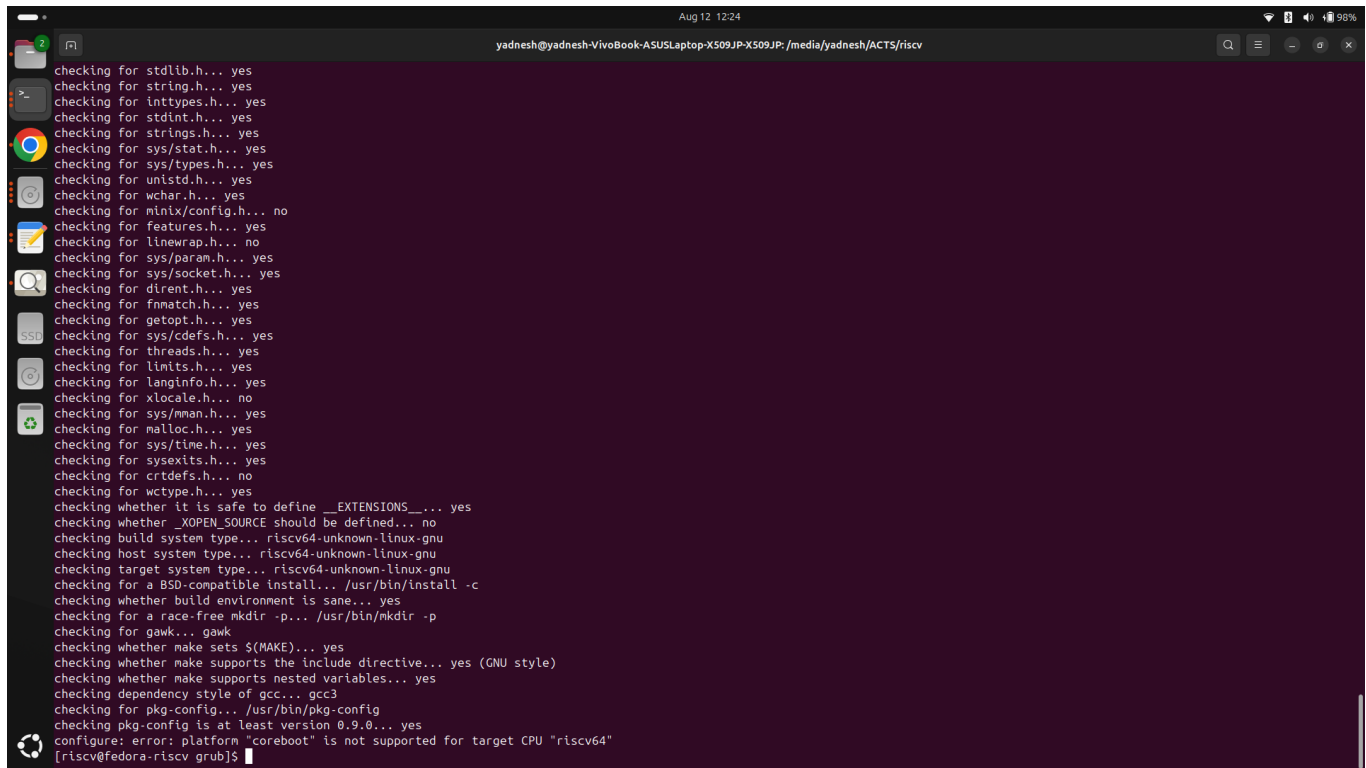
Fedora Linux 41 (Server Edition Prerelease)
Kernel 6.11.0-0.rc1.20240729gitdc1c8034e31b.16.fc41.x86_64 on an x86_64 (ttyS0)

Web console: https://localhost:9090/

localhost login: desd
Password:
Last login: Sat Aug 10 11:04:48 on ttyS0
[desd@localhost ~]$
```

Fig: Successfully login for i386

Exploring various open source boot loader project based on UEFI implementation for RISC-V architecture



```
Aug 12 12:24
yadnesh@yadnesh-VivoBook-ASUSLaptop-X509JP-X509JP: /media/yadnesh/ACTS/riscv
checking for stdlib.h... yes
checking for string.h... yes
checking for inttypes.h... yes
checking for stdint.h... yes
checking for strings.h... yes
checking for sys/stat.h... yes
checking for sys/types.h... yes
checking forunistd.h... yes
checking for wchar.h... yes
checking for minix/config.h... no
checking for features.h... yes
checking for linewrap.h... no
checking for sys/param.h... yes
checking for sys/socket.h... yes
checking for dirent.h... yes
checking for fnmatch.h... yes
checking for getopt.h... yes
checking for sys/cdefs.h... yes
checking for threads.h... yes
checking for limits.h... yes
checking for langinfo.h... yes
checking for xlocale.h... no
checking for sys/mman.h... yes
checking for malloc.h... yes
checking for sys/time.h... yes
checking for sys/exits.h... yes
checking for crtdefs.h... no
checking for wctype.h... yes
checking whether it is safe to define __EXTENSIONS__... yes
checking whether _XOPEN_SOURCE should be defined... no
checking build system type... riscv64-unknown-linux-gnu
checking host system type... riscv64-unknown-linux-gnu
checking target system type... riscv64-unknown-linux-gnu
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
checking for a race-free mkdir -p... /usr/bin/mkdir -p
checking for gawk... gawk
checking whether make sets $(MAKE)... yes
checking whether make supports the include directive... yes (GNU style)
checking whether make supports nested variables... yes
checking dependency style of gcc... gcc3
checking for pkg-config... /usr/bin/pkg-config
checking pkg-config is at least version 0.9.0... yes
configure: error: platform "coreboot" is not supported for target CPU "riscv64"
[riscv@fedora-riscv grub]$
```

Fig:

RISC-V

login

window

Chapter 6

Conclusion

6.1 Conclusion

Throughout this project, we have meticulously executed a series of tasks to enhance the deployment capabilities of Linux on RISC-V architecture, focusing on future-generation embedded systems. Our journey began with the compilation of COREBOOT for RISC-V with Grub support, ensuring compatibility and efficient booting across a variety of RISC-V devices. This step was crucial in establishing a solid foundation for our deployment solution.

Subsequently, we compiled Coreboot and seamlessly integrated it with Coreboot, providing

a robust runtime environment essential for managing software on RISC-V platforms. This integration not only enhanced system stability but also ensured efficient initialization and management of software, contributing significantly to the overall performance of our deployment solution.

A major milestone of our project was the compilation of the Linux kernel with support for containers and virtualization, a pivotal step in enabling the deployment and management of containerized applications on RISC-V devices. This enhancement opens up a myriad of possibilities for developers, allowing them to leverage containerization technologies to create more efficient and scalable applications for RISC-V platforms.

Additionally, we integrated a Fedora-based root file system managed , simplifying the deployment process of software packages on RISC-V devices. This integration not only streamlines deployment workflows but also ensures compatibility and ease of use for developers working with RISC-V architecture.

Furthermore, we successfully built a Fedora image for RISC-V using the QEMU emulator, allowing for thorough testing and validation of our deployment solution. This comprehensive approach to image composition and integration ensures that our deployment solution meets the stringent requirements of future-generation embedded systems running on RISC-V architecture.

Impact and Significance:

The significance of our project extends beyond its immediate outcomes, offering a substantial contribution to the advancement of deployment capabilities for Linux on RISC-V architecture. By providing optimized Coreboot and Fedora images, our project enhances deployment efficiency, scalability, and compatibility, addressing current limitations and fostering innovation in the RISC-V ecosystem. This contribution is poised to have a profound impact on the developer, researcher, and broader technological

communities engaged in RISC-V and Cotechnologies, catalysing the development of more sophisticated and reliable embedded systems solutions.

6.2 Future Enhancement –

While project has achieved its core objectives, there's room for further development:

- Implementing GRUB as bootloader for ease of use for RISC-V.
- Implementing more secured method by signing kernel and implementing secure boot.
- Upstream contributions: Sharing advancements and expertise with the wider Container community.

Chapter 7

References

[1] <https://fedoraproject.org/wiki>

[2] <https://riscv.org/>

[3] <https://www.gnu.org/software/grub/grub.html>

[4]<https://docs.kernel.org/>

[5]<https://www.coreboot.org/Payloads>

[6] <https://doc.coreboot.org/>.

[7] <https://www.coreboot.org/>.