HOSTED BY

Contents lists available at ScienceDirect

Engineering Science and Technology, an International Journal

journal homepage: www.elsevier.com/locate/jestch

Preprint Submitted to Elsevier

Design and Implementation of Secure Boot Architecture on RISC-V using FPGA

Loo, Tung Lun[a], Mohamad Khairi Ishak[b]

[a] School of Electrical and Electronic Engineering, Universiti Sains Malaysia, 14300, Nibong Tebal, Pulau Pinang, Malaysia.
[b] School of Electrical and Electronic Engineering, Universiti Sains Malaysia, 14300, Nibong Tebal, Pulau Pinang, Malaysia.

ARTICLE INFO

Article history:
Received 27 July 2021
Revised xx Month 2021
Accepted xx Month 2021
Available online xx Month 2021

Keywords:
RISC-V
Security
Firmware

ABSTRACT

There are many well-known open-source bootloaders solutions available today such as UEFI/BIOS, Coreboot and Uboot. Recently, RISC-V as an open-source Instruction Set Architecture, has gained a lot of attention in new embedded products creation and academic research purpose. In this study, x86, ARM and RISC-V Instruction Set Architecture boot flow and boot solutions are studied, simulated, experimented, and summarized. Security feature is implemented in firmware and measured against non-secured firmware to compare boot performance without security inclusion. A new proposed method to create a security block in Register Transfer Level to generate Secure Hash Algorithms 5 digest is implemented using Field Programmable Gate Array. The performance of this method is analyzed with the numbers of logic gate required and the execution time in software versus hardware. As a result of this study, it is observed that in simulated environment, secured firmware incurred 3.3 Megabytes of additional binary size and 747ms (35 %) additional boot time compared to non-secured firmware. A hardware implementation is proposed in Field Programmable Gate Array (FPGA) to reduce the need for a larger size firmware and longer boot time to implement security. The results of this implementation indicate a requirement of 32,048 gates to implement a SHA512 IP that reduce software execution time by 1132 %.

1. Introduction

All compute devices today are powered by a few processors Instruction Set Architectures (ISAs), predominantly x86, AMD, ARM, and MIPS which is later converged to RISC-V in 2021 [1]. These ISAs provide flexibilities and extensibilities to the different engineering audiences, creating tremendous opportunities today that benefits consumer in many custom applications and use cases, especially in the booming edge devices in Internet of Things world. While having multiple ISA options are good, it is often difficult to make a good decision on which architecture to go for, because there are many factors that contribute to design decision. Several key elements of consideration while picking an ISA are as below.

- Time-To-Market (TTM)

  The TTM factor is about how easy it is to enable an embedded system with collaterals provided by the ISA provider. For example, the development time of an engineering team (often called OEM/ODM) taking a new 11th Generation Intel chip and providing a full solution with it. Several key factors that directly impact TTM are the availabilities of documentation, system level open-source references and manufacturing technology.

- Cost

  This factor includes cost of licensing, software, and hardware development cost that the OEM/ODM needs to pay to get the products released.

- Design flexibilities

  The design flexibilities revolve around two key questions of “How easy it is to include a new custom IP in a new design?” and “How easy it is to land firmware, driver, and software support of a new IP?"

An ideal SOC would not only needs to be functional, but also be protected since the very early initialization flow to ensure no malicious code can be injected at any point before arriving at user space applications. To achieve this, firmware architecture becomes an important topic of exploration to identify the security scheme offered with different ISA and how a generic security approach can be deployed to implement security in each of them. The gap of today’s security scheme is the ease of deployment whereby the enablers and users would often end up disabling security just to

https://doi.org/xx.xxxx/j.jestch.20xx.xx.xxx

improve the performance of the system, reduce the TTM and product price. The consequence of this problem will be more unsecured devices being in the market, causing risks to everyone in the IOT chain. Therefore, this research will focus on identifying the boot elements of each ISA, methodology to enable secure boot, and how a security IP block can be added to the register transfer and firmware level to facilitate security such that it does not significantly jeopardize system performance and is easy to enable without much additional software development.

The key objective identified for this research is to evaluate the secure firmware feature, measure it against boot performance, and propose security enhancement through Field Programmable Gate Array (FPGA) for firmware booting mechanism with the evaluated security features of an open-source ISA. This enhancement could be potentially scale to close-source ISA.

In this paper, section 1 describes the introduction, problems and objectives of the study. Section 2 describes the background and previous work related to RISC-V processor, boot flow, and security. Section 3 describes the proposed method, which includes secure boot in Software (QEMU) and hardware implementation through FPGA. Section 4 describes the experimental setup results and discussion. Section 5 concludes the study and identify future improvement opportunities.

2. Background and previous work

2.1. RISC-V processor and security

TBD: Talk more about RISC-V..... Talk about the boot flow of RISC-V. Include ISA specifics, eg. section 2.4.3, 2.3.3

2.2. Related work

In the past, many researches related to security features available in different ISA had been carried out. Ning et al analyzed several hardware assisted TEE such as Intel SGX, ARM TrustZone Technology and AMD SEV, and evaluated the feasibility of deploying them on edge compute infrastructure by evaluating the performance overhead [2]. More security and performance benchmark study had been carried out by Christian et al and it has been discovered that AMD SEV has the best benchmark with memory protection mechanism at execution speed of near native speed compared to Intel SGX [3]. Other than performance evaluation, there are also comparative study on multiple ISAs. Geraldine et al summarized some key challenges of security features, short coming of ARM TrustZone and Intel SGX, and proposed countermeasures in RISC-V architecture that address its defined thread models [4]. In respective targeted technology domain, Pascal et al proposed HECTOR-V, a RISC-V based architecture to improve on the flexibilities of peripherals' permission management [5].Victor et al on the other hand, proposed Sanctum, a RISC-V based TEE to improve on software isolation with comparison to Intel SGX [6]. There are also several researches that provide deep dive study and survey on existing technologies, for example, Intel SGX was deeply explained from different aspects from the technology use cases to vulnerabilities by Victor et al [7], an ARM TrustZone comprehensive survey was carried out by Sandro et al [8] and multiple System Management Mode usage model for security purposes was analyzed by William in his PHD report [9].

Need to copy table 2.4 and 2.5.

3. Proposed Method

The approach of the research to propose the methodology and collect results is demonstrated in Figure 1. In section 2.1, the platform alternatives are compared and selected. In section 2.2, secure boot implementation is proposed for RISC-V with firmware implementation in software emulated environment (QEMU). In section 2.3, secure boot implementation is proposed for RISC-V with RTL implementation in FPGA hardware environment. In section 2.4, experiments and parameters are proposed to evaluate the firmware and RTL implementation.

3.1. Secure Boot in Software/QEMU

The flow to identify secure boot ingredient has two key flows. The difference between the 2 flows are demonstrated as Figure 3, whereby the firmware is first being compiled and executed without any secure boot ingredient, then showcase how inject secure boot code and execute the firmware. As the firmware code boot stages are compiled as separate binaries, each of the binaries can be then hashed and verified.

Figure 4 describes the potential to apply UEFI secure boot to the existing RISC-V boot with the existing UEFI framework on x86 QEMU. The idea is to inject security stack in UEFI PEI and DXE phase so that the RISCV UEFI boot flow can have secure boot encapsulated. This topic was flagged as a potential enabling item by RISC-V presentation [10]. The security stack by UEFI services had been made available with OpenSSL as the underlayer library and a comprehensive technical report of this describing how to sign and incorporate the keys has been created by [11].

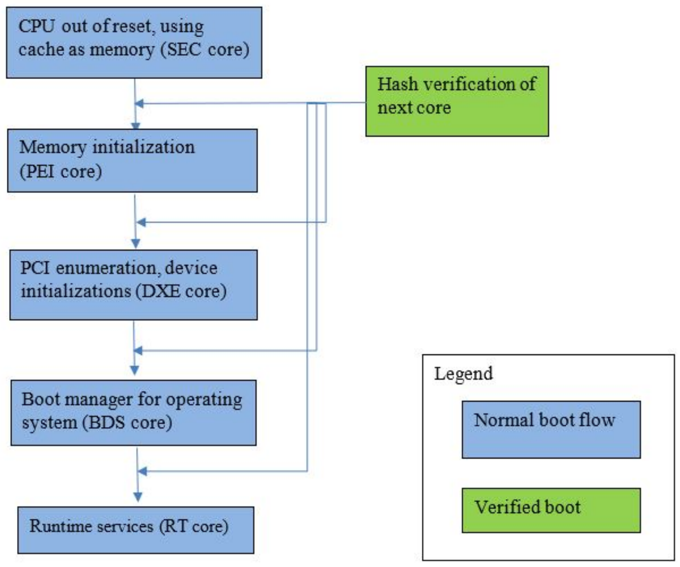

CPU out of reset, using cache as memory (SEC core)
Memory initialization (PEI core)
PCI enumeration, device initializations (DXE core)
Boot manager for operating system (BDS core)
Runtime services (RT core)
Hash verification of next core
Legend
Normal boot flow
Verified boot

Fig. 1. SoftwareSecureBootFlow

3.2. Hardware implementation through FPGA

Another part of the methodology is to propose a method to replace these secure boot services with RTL instead of bootloader code to effectively reduce flash size and improve boot performance. To achieve this, an open-source RISCV processor (NEORV32) is used as an initial environment. SHA512 digest generation block is then added to the RTL of NEORV32, and the digest generated is passed to bootloader via a new read only register block through

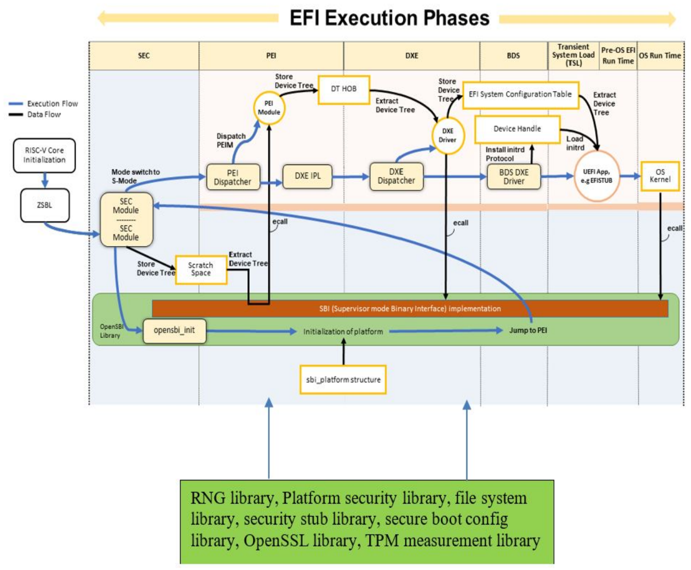


Fig. 2. UEFI Secured Boot on RISC-V

the custom functions subsystem (CFS) IP. With this implementation, bootloader will no longer need to contain and execute security code to achieve security purpose. A bird-eye view of what is being added is illustrated in green boxes of Figure 5. The IP are customized to introduce an additional arbitration block with state machine that is capable to map the boot rom content to be sent to SHA512 security block to produce digest. Once digest data is produced, the arbitration block will notify custom functions subsystem block with a status complete bit together with 512-bit SHA information.

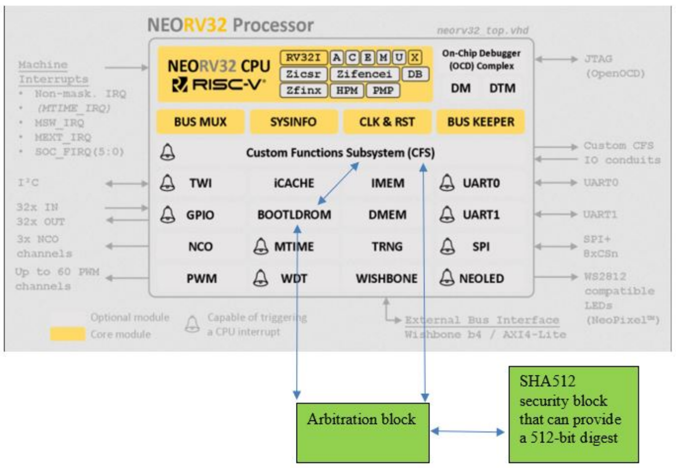


Fig. 3. Hardware Secure Boot Block

Figure 6 and figure 7 demonstrates the detailed comparison of system port map after Arbiter and SHA512 core is being added to NEOV32 CPU processor. The details of how the SHA512 core and arbiter block from signals level and how they are being consumed is described subsequently.

Based on the connections shown in Figure 7, during normal boot up process, NEORV32 CPU fetches instructions from bootloader ROM to execute. The amount of memory mapped IO and functionalities depend heavily on how the CPU is connected to data bus and in this case, the custom functions system block. The custom functions system block defines an interface consist of offset of each data that bootloader can read and write data from. The SHA512 Core block is responsible for taking in blocks of data to hash, and then provide output of the digest once completed. Arbiter is the middleman which controls the operation of taking ROM data and send to SHA512 Core to be hashed. Once the hash operation is completed, the digest and the completion status will then be shared with custom functions system to be accessible by CPU, which translate to software accessible registers.

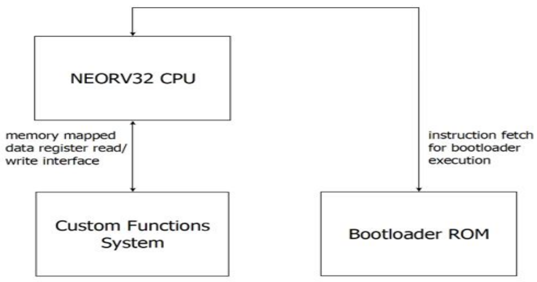

NEORV32 CPU
memory mapped data register read/ write interface
instruction fetch for bootloader execution
Custom Functions System
Bootloader ROM

Fig. 4. System port map of components without security block

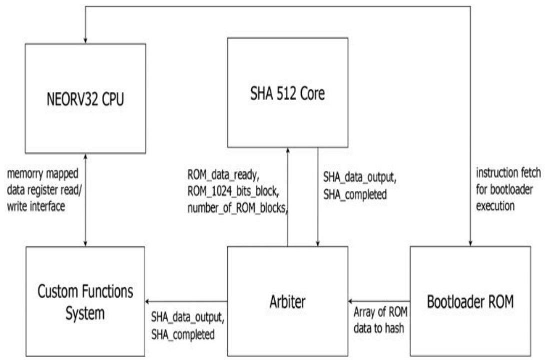

NEORV32 CPU
SHA 512 Core
memory mapped data register read/ write interface
ROM_data_ready, ROM_1024_bits_block, number_of_ROM_blocks,
SHA_data_output, SHA_completed
instruction fetch for bootloader execution
Custom Functions System
SHA_data_output, SHA_completed
Arbiter
Array of ROM data to hash
Bootloader ROM

Fig. 5. System port map of components to implement additional security block

The state machine of the arbiter is further designed as Figure 8. It begins with Idle state when everything is initialized to 0. A counter is implemented to keep track of the ROM blocks left to transfer from ROM block to SHA512 core. In send data state, the arbiter will set ROM_data_ready for SHA512 block to consume that block of 1024 bits data, then transition to toggle data ready bit state to toggle ROM_data_ready bit to 0 and decrement the counter so that the next send data state will transfer next chunk of data to SHA512 core to be processed. Once all blocks had been sent to SHA512 core. It will wait for SHA512 block to respond with SHA_complete. Once SHA_complete is set to 1, it will enter a complete state and update the SHA data to custom functions system block together with the SHA_complete status bit.

3.3. Proposed experiments

The experiment proposed to evaluate the methodology includes 3 parameters.

1. Boot performance (time used to boot the firmware)
2. Firmware size (size of the compiled binary)
3. Number of logic gates consumed in RTL (FPGA resource consumed)

To prove secured boot implementation, negative testing will be performed using unauthorized EFI applications that mimics malicious software to demonstrate that only signed applications can

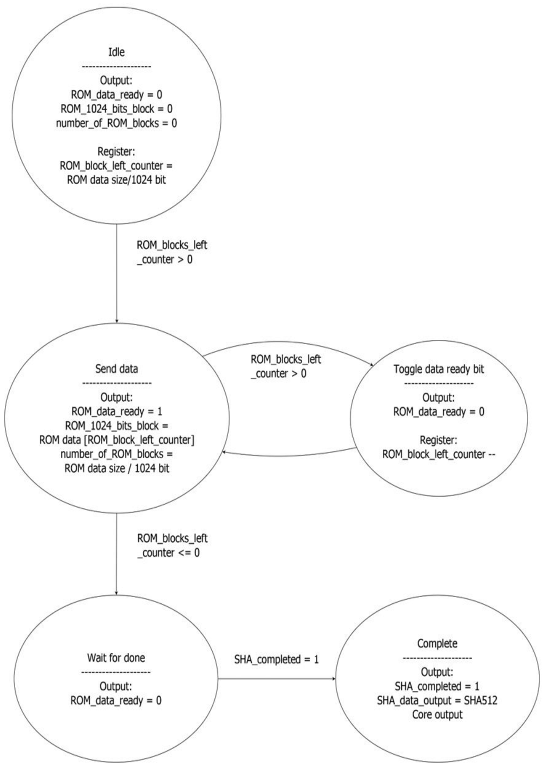


Fig. 6. State machine of arbitration block

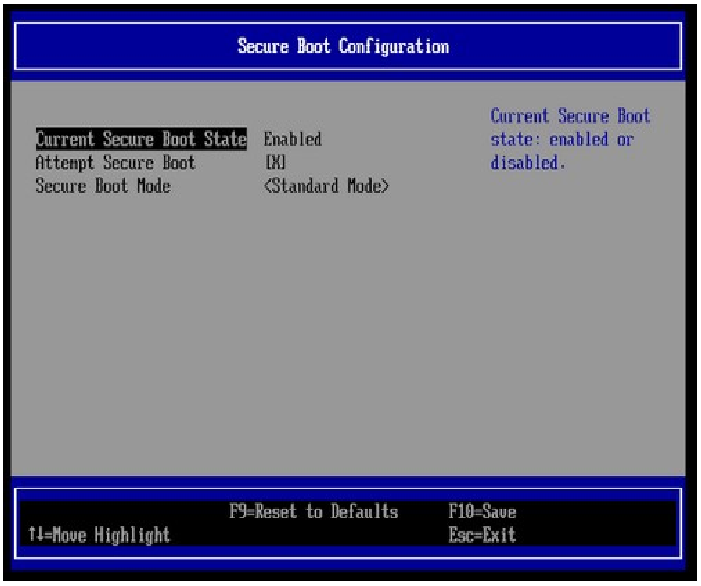
be executed. Therefore, the flow of experiments for software and hardware is planned respectively.

For Software QEMU, a normal unsecured UEFI firmware for x86 and RISC-V is compiled. Then, the UEFI firmware in QEMU is executed by booting to Shell and capturing the boot log. After this is achieved, a secured UEFI firmware is compiled. Then, the secured UEFI firmware in QEMU is executed by booting to Shell and capturing the boot log. With the secured UEFI firmware, negative testing is performeced with signed and unsigned EFI application. The boot time of secured and unsecured firmware are captured. With this and by comparing the results, the impact to boot time after incorporated security can be benchmarked. The binary size of secured and unsecured firmware are captured. With this and by comparing the results, the impact to firmware binary size after incorporated security can be benchmarked.

For Hardware FPGA, the SHA512 block that interacts with other components in NEORV32 RISC-V processor is implemented. The SHA512 digest generated by the SHA512 block is captured and compare with the output of software execution to verify the correctness in functionality. The SHA512 digest generation time is captured to be compared with software execution time to verify the performance. The boot log of NEORV32 firmware is captured to verify the ability to access SHA512 digest generated by RTL in bootloaders.

4. Results/Discussion

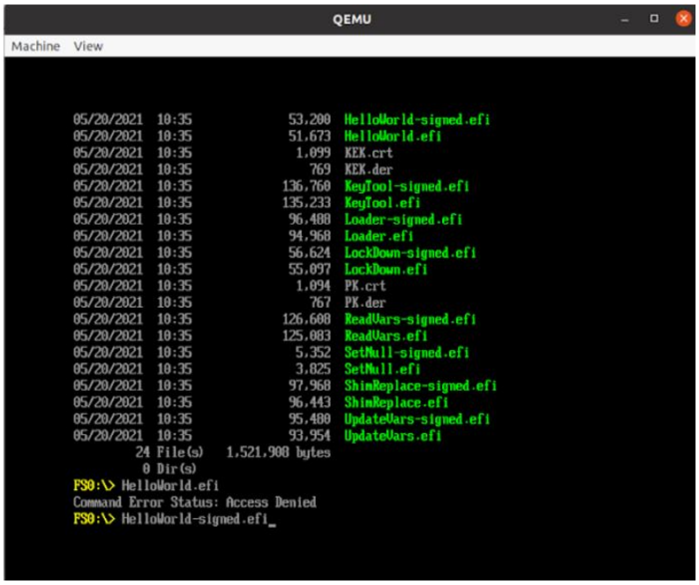
From functional correctness perspective, secured boot is configured as Figure 9. With secure boot enabled, only application software that is being signed with the same keys will be able to execute. To validate this behavior, an unsigned “Hello World” application and a signed “Hello World” application is attempted to execute in EFI Shell environment. Figure 10 shows the unsigned application and Figure 11 shows the signed application. The commit ID used to produce this result is 392836a for efitools repository and 75e9154f81 for EDK2 repository.


```
                         Secure Boot Configuration

 Current Secure Boot State   Enabled                  Current Secure Boot
 Attempt Secure Boot         [X]                      state: enabled or
 Secure Boot Mode            <Standard Mode>          disabled.

                  F9=Reset to Defaults       F10=Save
 ↑↓=Move Highlight                           Esc=Exit
```

Fig. 7. Secure boot configuration



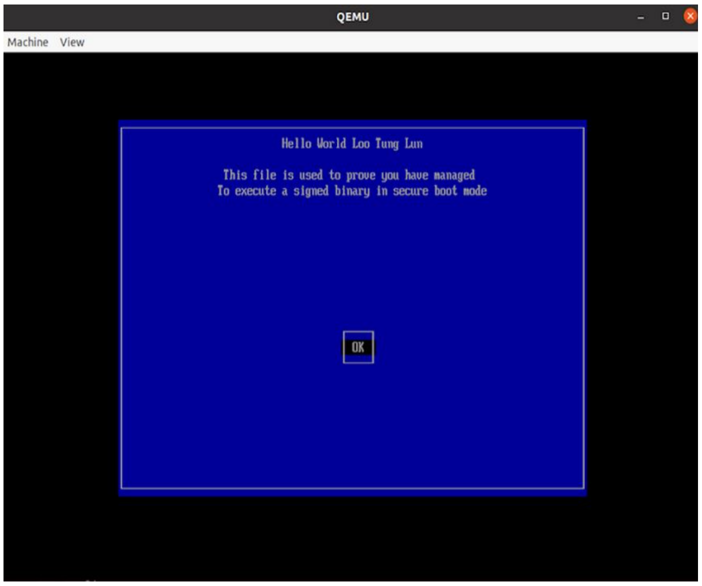
```
05/20/2021  10:35            53,200  HelloWorld-signed.efi
05/20/2021  10:35            51,673  HelloWorld.efi
05/20/2021  10:35             1,099  KEK.crt
05/20/2021  10:35               769  KEK.der
05/20/2021  10:35           136,760  KeyTool-signed.efi
05/20/2021  10:35           135,233  KeyTool.efi
05/20/2021  10:35            96,488  Loader-signed.efi
05/20/2021  10:35            94,968  Loader.efi
05/20/2021  10:35            56,624  LockDown-signed.efi
05/20/2021  10:35            55,097  LockDown.efi
05/20/2021  10:35             1,094  PK.crt
05/20/2021  10:35               767  PK.der
05/20/2021  10:35           126,608  ReadVars-signed.efi
05/20/2021  10:35           125,083  ReadVars.efi
05/20/2021  10:35             5,352  SetNull-signed.efi
05/20/2021  10:35             3,825  SetNull.efi
05/20/2021  10:35            97,968  ShimReplace-signed.efi
05/20/2021  10:35            96,443  ShimReplace.efi
05/20/2021  10:35            95,480  UpdateVars-signed.efi
05/20/2021  10:35            93,954  UpdateVars.efi
          24 File(s)   1,521,908 bytes
           0 Dir(s)
FS0:\> HelloWorld.efi
Command Error Status: Access Denied
FS0:\> HelloWorld-signed.efi_
```

Fig. 8. Load unsigned "Hello World" application

From firmware size perspective, results collected indicates that non-secure UEFI firmware has a total of 7,602,384 bytes, while secure UEFI firmware has a total of 10,895,864 bytes. Therefore, it is deduced that from software perspective, implementing security in firmware will add additional 3.292 Megabytes (10.895M – 7.602M) of additional binary size.

In terms of boot speed, it is observed that non-secure firmware took 2092 milliseconds to boot while secure firmware took 2839 milliseconds to boot in QEMU. Therefore, it is deduced that there are an additional 747 milliseconds (2839ms – 2092ms) additional boot time that secured firmware has in extra, compared to non-secured firmware, which is 747ms/2092ms * 100 = 35.7%.

From hardware perspective, results collected indicates that a RISC-V based NEORV32 without any additional security implementation will consume 19,785 logic gates, while a NEORV32



Hello World Loo Tung Lun

This file is used to prove you have managed
To execute a signed binary in secure boot mode

Fig. 9. Loading signed "Hello World" application

Table 1
Hardware and Software Performance Comparison

	Execution Time with 2.2GHz frequency CPU using same set of data
Software execution	257 us
Hardware execution	227 ns

with the addition of arbiter and SHA512 block consumes 51,833 logic gates. Therefore, it is deduced that implementing security in RTL will add 32,048 logic gates. In terms of security execution speed comparison, it is observed that producing a SHA512 digest for 896 bits data will take 257us with software while RTL implementation takes 227ns. The performance advantage is therefore 257u/227n * 100 = 1132%.

5. Conclusion

The objective of this research, which is to study each ISA security schemes, identify and evaluate boot performance with different secure boot scheme, and propose a security enhancement mechanism with open-source ISA, is accomplished. The boot time and boot size impact of implementing hashing for firmware is highlighted in Slim Bootloader verified boot comparison, which indicates that secured firmware incurred 3.3 Megabytes of additional binary size and 747ms (35%) additional boot time compared to non-secured firmware. The hardware implementation also indicates that it requires an additional 32,048 logic gates to implement a SHA512 IP that reduce software execution time by 1132%.

Although this paper has demonstrated the secure boot implementations with QEMU and FPGA hardware, there are some enhancement to be done to enable security with minimal firmware or software involvement, driven by the initial problem statement. One example is how the configuration to update the RTL security scheme at runtime can be provided for better user experience. A suggestion is through a network IP with manageability mechanism for Over-The-Air (OTA) update, connecting with RISC-V network on chip cores, such as the OpenPiton Network on Chip (NOC) project and have a secure channel to modify the key hashes. This is another topic of research area that can be proposed and presented with industrial use cases with business opportunities to introduce such features on IOT secured devices.

Acknowledgment

The authors would like to thank USM for the support and Intel for the sponsorship.

References

[1] J. Turley, “Wait, what? mips becomes risc-v.” https://www.eejournal.com/article/wait-what-mips-becomes-risc-v/, March 2021. Accessed on 2021-8-30.

[2] Z. Ning, J. Liao, F. Zhang, and W. Shi, “Preliminary study of trusted execution environments on heterogeneous edge platforms,” *Third ACM/IEEE Symposium on Edge Computing*, 2018.

[3] C. Göttel., R. Pires., I. Rocha., S. Vaucher., P. Felber., and M. P. V. Schiavoni, “Security, performance and energy trade-offs of hardware-assisted memory protection mechanisms,” *2018 IEEE 37th Symposium on Reliable Distributed Systems (SRDS), Salvador, Brazil*, 2019.

[4] N. G. Shirley., G. Yutian., and S. Fareena, “A survey and analysis on soc platform security in arm, intel and risc-v architecture,” *2020 IEEE 63rd International Midwest Symposium on Circuits and Systems (MWSCAS)*, 2020.

[5] P. Nasahl., R. Schilling., M. Werner., and S. Mangard, “Hector-v: A heterogeneous cpu architecture for a secure risc-v execution environment,” *2021 ACM Asia Conference on Computer and Communications Security*, 2021.

[6] V. Costan., I. Lebedev., S. Devadas., and M. CSAIL, “Sanctum: Minimal hardware extensions for strong software isolation,” *USENIX The Advanced Computing System Association*, 2016.

[7] V. Costan. and S. Devadas, “Intel sgx explained,” *Computer Science and Artificial Intelligence Laboratory Massachusetts Institute of Technology*, 2016.

[8] S. Pinti. and N. Santos, “Demystifying arm trustzone: A comprehensive survey,” *ACM Computing Survey Volume 51, Issue 6*, 2019.

[9] d. S. William Augusto Rodrigues, “On using the system management mode for security purpose,” *University of London, Doctor of Philosophy*, 2016.

[10] D. Schaefer, “Edk2 uefi on risc-v.” https://fosdem.org/2021/schedule/event/firmware_uor/attachments/slides/4492/export/events/attachments/firmware_uor/slides/4492/EDK2_on_RISC_V.pdf, February 2021. Accessed on 2021-8-30.

[11] N. S. Agency, “Uefi secure boot customization.” https://media.defense.gov/2020/Sep/15/2002497594/-1/-1/0/CTR-UEFI-SECURE-BOOT-CUSTOMIZATION-20200915.PDF/CTR-UEFI-SECURE-BOOT-CUSTOMIZATION-20200915.PDF, September 2020. Accessed on 2021-8-30.