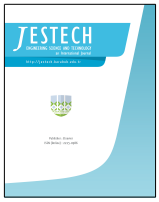
HOSTED BY

ELSEVIER

Contents lists available at ScienceDirect

Engineering Science and Technology, an International Journal

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



Preprint Submitted to Elsevier

A Simulated Firmware Study on X86, ARM, RISC-V Platform and Security Enhancement For RISC-V

Loo, Tung Lun, Mohamad Khairi Ishak

[a] *Sungai Petani, 08000 Kedah, Malaysia*
[b] *USM, 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 (Jim Turley, EE Journal, 2021). 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. 1. 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.2 2. Cost This factor includes cost of licensing, software, and hardware development cost that the OEM/ODM needs to pay to get the products released. 3. 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?"

1.1. Problem Statements

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 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.

https://doi.org/xx.xxxx/j.jestch.20xx.xx.xxx
2215-0986/

1.2. Objectives

With the problem statement described, three key objectives are identified for this research. 1. To simulate each ISAs boot flow and identify the key components that differentiate each of them from the firmware developers' perspective. 2. To evaluate the secure firmware feature and measure it against boot performance. 3. To propose and implement 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.

2. Proposed Method

The workflow to propose the methodology for this research is demonstrated in Figure 3.1. In section 3.2, the platform alternatives are compared and selected. In section 3.3, secure boot implementation is proposed for RISC-V with firmware implementation in software emulated environment (QEMU). In section 3.4, secure boot implementation is proposed for RISC-V with RTL implementation in FPGA hardware environment. In section 3.5, experiments and parameters are proposed to evaluate the firmware and RTL implementation. In section 3.6, the chapter is summarized.

2.1. Platform Selection

A comparison between these platforms is created, considering the factor of cost, scalability, availability of resource, and configurability as illustrated in Table 3.1

2.2. Secure Boot in Software/QEMU

The flow to identify secure boot ingredient has two key steps. Both steps are demonstrated as Figure 3.3, 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. The general boot flow demonstrated uses SEC core, PEI core, DXE core and BDS core as demonstrated by Figure 2.1. As each core are compiled as separate binaries, each of the binaries can be hashed and verified.

Based on the flow described in figure 3.2, the potential to apply UEFI secure boot to the existing RISC-V boot with UEFI framework on x86 QEMU is identified. 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 (Schaefer, 2021). 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 (National Security Agency, 2020). The security driver injection in the common UEFI services FDF and DSC file (Jarlstrom, 2017) is demonstrated as Figure 3.4.

2.3. 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 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 3.5. 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.

Figure 3.6 and figure 3.7 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 3.7, 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.

3. Results/Discussion

In section 4.2, 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. 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%.

In section 4.3, 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 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%.

4. Conclusion

The first objective of this research, which is to study each ISA and key boot factors in details, is accomplished and outlined in the comprehensive literature review describing boot flow, secure firmware solutions and RISC-V projects.

The second objective, which is to identify boot performance and evaluation with different secure boot scheme and performance, is accomplished with experiments carried out with x86 QEMU and RISC-V QEMU, along with RISC-V secure boot enable activities in

methodology. The boot time impact of hashing for firmware is also highlighted in Slim Bootloader verified boot capture. It is observed that secured firmware incurred 3.3 Megabytes of additional binary size and 747ms (35%) additional boot time compared to non-secured firmware.

The third objective, which is to propose a security enhancement mechanism with open-source ISA, is accomplished by the RISC-V SHA512 secure boot enablement carried out with Xilinx FPGA hardware. Based on the results, it is observed that the results of this implementation indicate a requirement of additional 32,048 logic gates to implement a SHA512 IP that reduce software execution time by 1132%.

4.0.1. Future Work

Although this thesis has demonstrated the secure boot implementations with QEMU and FPGA hardware, the work to enable security with minimal firmware or software involvement does not stop here. The problem statement of reducing ROM size and boot performance will drive multiple areas of enhancement. One example is how the configuration to update the RTL security scheme at runtime can be provided for better user experience. A good suggestion is through a IOT 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 Internet of Things secured devices.

Acknowledgment

The authors would like to thank the editors and anonymous reviewers for providing insightful suggestions and comments to improve the quality of research paper.